ENS contest
Findings & Analysis Report

2022-12-13

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 contest 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 contest outlined in this document, C4 conducted an analysis of the ENS smart contract system written in Solidity. The audit contest took place between July 12—July 19 2022.

Wardens

107 Wardens contributed reports to the ENS contest:

1. PwnedNoMore (izhuer, ItsNio, papr1ka2, and wen)
2. zzzitron
3. panprog

4. GimelSec (rayn and sces60107)
5. 0x52
6. alan724
7. csanuragjain
8. 0x1f8b
9. wastewa
10. Aussie_Battlers (sseefried and oyc_109)
11. brgltd
12. cryptphi
13. peritoflores
14. cccz
15. Lambda
16. llllllll
17. Dravee
18. bin2chen
19. 0x29A (0x4non and rotcivegaf)
20. Limbooo
21. ronnyx2017
22. joestakey
23. rbserver
24. 0xKitsune
25. benbaessler
26. Sm4rty
27. berndartmueller
28. Bnke0x0
29. Deivitto
30. RedOneN
31. CRYP70
32. gogo

33. Amithuddar
34. hake
35. TomJ
36. MiloTruck
37. Rolezn
38. c3phas
39. Ruhum
40. Ch_301
41. _Adam
42. hyh
43. __141345__
44. asutorufos
45. fatherOfBlocks
46. 0xNineDec
47. rajatbeladiya
48. 0xNazgul
49. robee
50. sashik_eth
51. Funen
52. kyteg
53. Waze
54. JC
55. JohnSmith
56. Rohan16
57. bulej93
58. cRat1st0s
59. rokinot
60. delfin454000
61. 8olidity

62. sach1r0
63. ReyAdmirado
64. zuhaibmohd
65. lcfr_eth
66. simon135
67. seyni
68. cryptonue
69. ElKu
70. dxdv
71. pashov
72. 0xf15ers (remora and twojoy)
73. p_crypt0
74. Critical
75. pedr02b2
76. philogy
77. 0xDjango
78. rishabh
79. svskaushik
80. RustyRabbit
81. minhtrng
82. exd0tpy
83. m_Rassska
84. durianSausage
85. ajtra
86. Tomio
87. 0x040
88. 0xsam
89. Aymen0909
90. Fitraldys

91. lucacez
92. Noah3o6
93. samruna
94. arcoun
95. karanctf
96. sahar
97. ak1
98. Chom
99. Jujic
100. scaraven

This contest was judged by LSDan.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 16 unique vulnerabilities. Of these vulnerabilities, 3 received a risk rating in the category of HIGH severity and 13 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 71 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 70 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 contest repository, and is composed of 23 smart contracts written in the Solidity programming language and includes 2,132 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into 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

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings (3)

[H-01] It is possible to create fake ERC1155 `NameWrapper` token for subdomain, which is not owned by `NameWrapper`

Submitted by panprog, also found by Aussie_Battlers, brgltd, cryptphi, peritoflores, and wastewa

NameWrapper.sol#L820-L821
NameWrapper.sol#L524
NameWrapper.sol#L572

Due to re-entrancy possibility in `NameWrapper._transferAndBurnFuses` (called from `setSubnodeOwner` and `setSubnodeRecord`), it is possible to do some stuff in `onERC1155Received` right after transfer but before new owner and new fuses are set. This makes it possible, for example, to unwrap the subdomain, but owner and fuses will still be set even for unwrapped domain, creating fake `ERC1155 NameWrapper` token for domain, which is not owned by `NameWrapper`.

Fake token creation scenario:

1. `Account1` registers and wraps `test.eth` domain
2. `Account1` calls `NameWrapper.setSubnodeOwner` for `sub.test.eth` subdomain with `Account1` as owner (to make NameWrapper owner of subdomain)
3. `Contract1` smart contract is created, which calls unwrap in its `onERC1155Received` function, and a function to send `sub.test.eth` ERC1155 NameWrapper token back to `Account1`
4. `Account1` calls `NameWrapper.setSubnodeOwner` for `sub.test.eth` with `Contract1` as new owner, which unwraps domain back to `Account1` but due to re-entrancy, NameWrapper sets fuses and ownership to `Contract1`
5. `Account1` calls function to send ERC1155 token from `Contract1` back to self.

After this sequence of events, `sub.test.eth` subdomain is owned by `Account1` both in `ENS` registry and in `NameWrapper` (with fuses and expiry correctly set to the future date). Lots (but not all) of functions in `NameWrapper` will fail to execute for this subdomain, because they expect `NameWrapper` to have ownership of the domain in `ENS`, but some functions will still work, making it possible to make the impression of good domain.

At this point, ownership in `NameWrapper` is “detached” from ownership in `ENS` and `Account1` can do all kinds of malcious stuff with its ERC1155 token. For example:

1. Sell subdomain to the other user, transfering `ERC1155` to that user and burning `PARENT_CANNOT_CONTROL` to create impression that he can’t control the domain. After receiving the payment, `Account1` can wrap the domain again, which burns existing ownership record and replaces with the new one with clear fuses and `Account1` ownership, effectively stealing domain back from unsuspecting user, who thought that `ERC1155` gives him the right to the domain (and didn’t expect that parent can clear fuses when `PARENT_CANNOT_CONTROL` is set).
2. Transfer subdomain to some other smart contract, which implements `onERC1155Received`, then take it back, fooling smart contract into believing that it has received the domain.

Proof of Concept

Copy these to test/wrapper and run:
yarn test test/wrapper/NameWrapperReentrancy.js

https://gist.github.com/panprog/3cd94e3fbb0c52410a4c6609e55b863e

Recommended Mitigation Steps

Consider adding `nonReentrant` modifiers with `ReentrancyGuard` implementation from `openzeppelin`. Alternatively just fix this individual re-entrancy issue. There are multiple ways to fix it depending on expected behaviour, for example saving `ERC1155` data and requiring it to match the data after transfer (restricting `onERC1155Received` to not change any data for the token received):

```
    function _transferAndBurnFuses(
        bytes32 node,
        address newOwner,
        uint32 fuses,
        uint64 expiry
    ) internal {
        (address owner, uint32 saveFuses, uint64 saveExpiry) = getDa
        _transfer(owner, newOwner, uint256(node), 1, "");
        uint32 curFuses;
        uint64 curExpiry;
        (owner, curFuses, curExpiry) = getData(uint256(node));
        require(owner == newOwner && saveFuses == curFuses && saveEx
        _setFuses(node, newOwner, fuses, expiry);
    }
```

Arachnid (ENS) confirmed

[H-02] The expiry of the parent node can be smaller than the one of a child node, violating the guarantee policy

Submitted by PwnedNoMore

NameWrapper.sol#L504
NameWrapper.sol#L356

By design, the child node's expiry can only be extended up to the parent's current one. Adding these restrictions means that the ENS users only have to look at the name itself's fuses and expiry (without traversing the hierarchy) to understand what guarantees the users have.

When a parent node tries to `setSubnodeOwner` / `setSubnodeRecord`, the following code is used to guarantee that the new expiry can only be extended up to the current one.

```
function _getDataAndNormaliseExpiry(
    bytes32 parentNode,
    bytes32 node,
    uint64 expiry
)
    internal
    view
    returns (
        address owner,
        uint32 fuses,
        uint64
    )
{
    uint64 oldExpiry;
    (owner, fuses, oldExpiry) = getData(uint256(node));
    (, , uint64 maxExpiry) = getData(uint256(parentNode));
    expiry = _normaliseExpiry(expiry, oldExpiry, maxExpiry);
    return (owner, fuses, expiry);
}
```

However, the problem shows when

- The sub-domain (e.g., `sub1.base.eth`) has its own sub-sub-domain (e.g., `sub2.sub1.base.eth`)
- The sub-domain is unwrapped later, and thus its `oldExpiry` becomes zero.
- When `base.eth` calls `NameWrapper.setSubnodeOwner`, there is not constraint of `sub1.base.eth`'s expiry, since `oldExpiry == 0`. As a result, the new expiry of `sub1.base.eth` can be arbitrary and smaller than the one of `sub2.sub1.base.eth`

The point here is that the `oldExpiry` will be set as 0 when unwrapping the node even it holds child nodes, relaxing the constraint.

Specifically, considering the following scenario

- The hacker owns a domain (or a 2LD), e.g., `base.eth`
- The hacker assigns a sub-domain to himself, e.g., `sub1.base.eth`
    - The expiry should be as large as possible
- Hacker assigns a sub-sub-domain, e.g., `sub2.sub1.base.eth`
    - The expiry should be as large as possible
- The hacker unwraps his sub-domain, i.e., `sub1.base.eth`
- The hacker re-wraps his sub-domain via `NameWrapper.setSubnodeOwner`
    - The expiry can be small than the one of sub2.sub1.base.eth

The root cause *seems* that we should not zero out the expiry when burning a node if the node holds any subnode.

Suggested Fix

- Potential fix 1: auto-burn `CANNOT_UNWRAP` which thus lets `expiry` decide whether a node can be unwrapped.
- Potential fix 2: force the parent to have `CANNOT_UNWRAP` burnt if they want to set expiries on a child via `setSubnodeOwner` / `setSubnodeRecord` / `setChildFuses`

Proof of Concept / Attack Scenario

For full details, please see original warden submission.

Arachnid (ENS) confirmed

[H-03] `PARENT_CANNOT_CONTROL` can be bypassed by maliciously unwrapping parent node

Submitted by PwnedNoMore, also found by panprog, and zzzitron

NameWrapper.sol#L356
NameWrapper.sol#L295
ENSRegistry.sol#L74

By design, for any subdomain, as long as its `PARENT_CANNOT_CONTROL` fuse is burnt (and does not expire), its parent should not be able to burn its fuses or change its owner.

However, this contraint can be bypassed by a parent node maliciously unwrapping itself. As long as the hacker becomes the ENS owner of the parent node, he can leverage `ENSRegistry::setSubnodeOwner` to re-set himself as the ENS owner of the subdomain, and thus re-invoking `NameWrapper.wrap` can rewrite the fuses and wrapper owner of the given subdoamin.

Considering the following attack scenario:

- Someone owns a domain (or a 2LD), e.g., *poc.eth*
- The domain owner assigns a sub-domain to the hacker, e.g., *hack.poc.eth*
    - This sub-domain should not burn `CANNOT_UNWRAP`
    - This sub-domain can burn `PARENT_CANNOT_CONTROL`
- Hacker assigns a sub-sub-domain to a victim user, e.g., *victim.hack.poc.eth*
- The victim user burns arbitrary fuses, including `PARENT_CANNOT_CONTROL`
    - The hacker should not be able to change the owner and the fuses of `victim.hack.poc.eth` ideally
- However, the hacker then unwraps his sub-domain, i.e., *hack.poc.eth*
- The hacker invokes `ENSRegistry::setSubnodeOwner(hacker.poc.eth, victim)` on the sub-sub-domain

    - He can reassign himself as the owner of the *victim.hack.poc.eth*
- The hacker invokes `NameWrapper.wrap(victim.hacker.poc.eth)` to over-write the fuses and owner of the sub-sub-domain, i.e., *victim.hacker.poc.eth*

The root cause here is that, for any node, when one of its subdomains burns `PARENT_CANNOT_CONTROL`, the node itself fails to burn `CANNOT_UNWRAP`. Theoretically, this should check to the root, which however is very gas-consuming.

Suggested Fix

- Potential fix 1: auto-burn `CANNOT_UNWRAP` which thus lets `expiry` decide whether a node can be unwrapped.
- Potential fix 2: leave fuses as is when unwrapping and re-wrapping, unless name expires. Meanwhile, check the old fuses even wrapping.

Proof of Concept / Attack Scenario

For full details, please see original warden submission.

Arachnid (ENS) confirmed

Medium Risk Findings (13)

[M-01] `wrapETH2LD` permissioning is over-extended

Submitted by 0x52

Undesired use of ENS wrapper.

Proof of Concept

NameWrapper.sol#L219-L223

Current permissioning for wrapETH2LD allows msg.senders who are not owner to call it if they are EITHER approved for all on the ERC721 registrar or approved on the wrapper. Allowing users who are approved for the ERC721 registrar makes sense. By giving them approval, you are giving them approval to do what they wish with the token. Any other restrictions are moot regardless because they could use approval to

transfer themselves the token anyways and bypass them as the new owner. The issue is allowing users who are approved for the wrapper contract to wrap the underlying domain. By giving approval to the contract the user should only be giving approval for the wrapped domains. As it is currently setup, once a user has given approval on the wrapper contract they have essentially given approval for every domain, wrapped or unwrapped, because any unwrapped domain can be wrapped and taken control of. This is an over-extension of approval which should be limited to the tokens managed by the wrapper contract and not extend to unwrapped domains

Recommended Mitigation Steps

Remove L221.

Arachnid (ENS) disagreed with severity and commented:

> This was by design, but the warden raises a good point about the implications of this permission model. Recommend downgrading to QA.

LSDan (judge) decreased severity to Medium and commented:

> I'm going to downgrade this to medium. There are not assets at direct risk, but with external factors the assets could be at risk due to the user being unaware that in approving wrapped domains, they are also approving unwrapped domains.

[M-02] Renew of 2nd level domain is not done properly

Submitted by csanuragjain, also found by cccz and GimelSec

ETHRegistrarController.sol#L201
NameWrapper.sol#L271

The ETHRegistrarController is calling renew from base registrar and not through Namewrapper. This means the fuses for the subdomain will not be updated via _setData. This impacts the permission model set over subdomain and could lead to takeover

Proof of Concept

1. Observe the renew function

```
function renew(string calldata name, uint256 duration)
        external
        payable
        override
    {
        ...

        uint256 expires = base.renew(uint256(label), duration);

        ....
    }
```

2. As we can see this is calling renew function of Base Registrar instead of NameWrapper. Since this is not going via NameWrapper fuses will not be set
3. Also since renew in NameWrapper can only be called via Controller which is ETHRegistrarController so there is no way to renew subdomain

Recommended Mitigation Steps

The ETHRegistrarController must renew using Namewrapper's renew contract.

Arachnid (ENS) commented:

> Duplicate of #223.

LSDan (judge) commented:

> On #223, which I've invalidated, @Arachnid notes that:
>
> In fact, this should only be severity QA, as it can be worked around by calling `renew` on the registrar controller followed by `setChildFuses`.
>
> I'm going to make this report the main one and leave the risk rating of Medium in place. While there is a workaround, if the workaround is not employed, permissions will be incorrect and may lead to a breakdown in the functioning of the protocol.

[M-03] `transfer()` depends on gas consts

Submitted by rajatbeladiya, also found by __141345__, _Adam, 0x29A, 0xNineDec, alan724, Amithuddar, asutorufos, Aussie_Battlers, berndartmueller, c3phas, cccz, Ch_301, cryptphi, csanuragjain, Dravee, durianSausage, fatherOfBlocks, GimelSec, hake, hyh, lllllll, Jujic, Limbooo, pashov, RedOneN, Ruhum, scaraven, TomJ, and zzzitron

[ETHRegistrarController.sol#L183-L185](#)
[ETHRegistrarController.sol#L204](#)

`transfer()` forwards 2300 gas only, which may not be enough in future if the recipient is a contract and gas costs change. it could break existing contracts functionality.

Proof of Concept

`.transfer` or `.send` method, only 2300 gas will be "forwarded" to fallback function. Specifically, the SLOAD instruction, will go from costing 200 gas to 800 gas.

If any smart contract has a functionality of register ens and it has fallback function which is making some state change in contract on ether receive, it could use more than 2300 gas and revert every transaction.

For reference, check out:

- https://docs.soliditylang.org/en/v0.8.15/security-considerations.html?highlight=transfer#sending-and-receiving-ether
- https://consensys.net/diligence/blog/2019/09/stop-using-soliditys-transfer-now/

Recommended Mitigation Steps

Use `.call` insted `.transfer`

```
(bool success, ) = msg.sender.call.value(amount)("");
require(success, "Transfer failed.");
```

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

> Recommend reducing severity to QA

LSDan (judge) decreased severity to Medium and commented:

> I'm downgrading this to Medium. There are external factors required to make this problem occur, but if it does the functionality of the protocol as a whole could be severely impacted.

Arachnid (ENS) commented:

> It's unclear to me how this could be a significant issue. Anyone writing code to register names knows that any excess funds will be returned, and therefore that they need a fallback that consumes minimal gas. Any EVM change that increases the gas of fallback functions would be breaking for a great number of contracts beyond ENS.

LSDan (judge) commented:

> It's unclear to me how this could be a significant issue.

> The register functions will fail, consuming a good bit of gas along the way if this occurs. If this were not such a critical piece of functionality I would have considered it QA, but a failure here breaks the protocol.

> Anyone writing code to register names knows that any excess funds will be returned, and therefore that they need a fallback that consumes minimal gas. Any EVM change that increases the gas of fallback functions would be breaking for a great number of contracts beyond ENS.

> The fact that other contracts will break along with ENS does not invalidate the issue. This is a clearly documented problem that has been known for years (see ref links). There is no reason to introduce more critical contracts to the ecosystem that will fail in this scenario, particularly when it is so easy to avoid.

Arachnid (ENS) commented:

> The register functions will fail, consuming a good bit of gas along the way if this occurs. If this were not such a critical piece of functionality I would have considered it QA, but a failure here breaks the protocol.

> I think the implied API here is that any contract registering names with the controller must either send the right amount of ether, or have a fallback function that can accept ether. Changes to the consensus layer that invalidate that assumption for a given contract are out-of-scope for ENS.

> Sending all remaining gas with the refund increases the threat surface by allowing possible reentrancy etc, which we haven't examined as a threat model here.

[M-04] `BytesUtil.compare` returns wrong result on some strings longer than 32 characters

Submitted by panprog, also found by alan724 and GimelSec

[BytesUtils.sol#L66-L70]

Due to incorrect condition in `ByteUtil.compare` function, irrelevant characters are masked out only for strings shorter than `32` characters. However, they must be masked out for strings of all lengths in the last pass of the loop (when remainder of the string is 32 characters or less). This leads to incorrect comparision of strings longer than `32` characters where `len` or `otherlen` is smaller than string length (characters beyond provided length are still accounted for in the comparision in this case while they should be ignored).

This wrong `compare` behaviour also makes `RRUtils.compareNames` fail to correctly compare DNS names in certain cases.

While the `BytesUtil.compare` and `RRUtils.compareNames` methods are currently not used in the functions in the scope (but are used in mainnet's `DNSSECImpl.checkNsecName`, which is out of scope here), they're public library functions relied upon and can be used by the other users or the ENS project in the future. And since the functions in scope provide incorrect result, that's a wrong (unexpected) behaviour of the smart contract. Moreover, since the problem can be seen only with the large strings (more than `32` characters), this might go unnoticed

with any code that uses `compare` or `compareNames` method and can potentially lead to high security risk of any integration project or ENS itself.

Example strings to compare which give incorrect result:

```
0123456789012345012345678901234 5ab
0123456789012345012345678901234 5aa
```

Each string is `34` characters long, first `33` characters are the same, the last one is different. If we compare first `33` characters of both strings, the result should be `equal` (as they only differ in the 34th character), but `compare` will return `>`, because it fails to ignore the last character of both strings and simply compares strings themselves.

If we compare the first `33` characters from the first string vs all `34` characters of the second string, the result of `compare` will be `>`, while the correct result is `<`, because `compare` fails to ignore the last character of the first string.

Example dns names to compare which give incorrect result:

```
0123456789012345678901234567890 1a0.0123456789012345678901234567890123456789012345678.eth
0123456789012345678901234567890 1a.0123456789012345678901234567890123456789012345678.eth
```

The first dns name should come after the second, but `dnsCompare` returns `-1` (the first name to come before), because the length of the 2nd domain (49 characters) is ASCII character `1` and is not correctly masked off during strings comparision.

Proof of Concept

git diff

https://gist.github.com/panprog/32adefdc853ccd0fd0f1aad85c526bea

then:

yarn test test/dnssec-oracle/TestSolidityTests.js

Recommended Mitigation Steps

In addition to the incorrect condition, the mask calculation formula: `32 - shortest + idx` will also overflow since `shortest` can be more than `32`, so addition should be performed before subtractions.

```
if (shortest - idx >= 32) {
    mask = type(uint256).max;
} else {
    mask = ~(2 ** (8 * (idx + 32 - shortest)) - 1);
}
```

Arachnid (ENS) confirmed

[M-05] `DNSSECImpl.verifySignature` compares strings incorrectly, allowing malicious zones to forge DNSSEC trust chain

Submitted by GimelSec, also found by csanuragjain

DNSSECImpl.sol#L186-L190

DNSSEC allows parent zones to sign for its child zones. To check validity of a signature, RFC4034 3.1.7 requires the `Signer's Name` in any RRSIG RDATA to contain the zone of covered RRset. This requirement is reasonable since any child zone should be covered by its parent zone.

ENS tries to implement the concept of name coverage in `DNSSECImpl.verifySignature`, but unfortuantely does it wrong, resulting in possibiliy of coverage between two unrelated domains. In the worst case, an attacker can utilize this bug to forge malicious trust chains and authenticate invalid domains.

Proof of Concept

In `DNSSECImpl.verifySignature`, ENS tries to verify the name of RRSet zone (`name`) is contained by Signer's Name (`rrset.signerName`).

```
        if(rrset.signerName.length > name.length
                || !rrset.signerName.equals(0, name, name.length - r
            {
                revert InvalidSignerName(name, rrset.signerName);
            }
```

In DNS, for a parent zone to contain another child zone, we generally require the child zone to be a subdomain of the parent. For instance, `example.eth.` in considered to cover `sub.example.eth.`, while `xample.eth.` should not be cover `example.eth.`.

Unfortunately in the implementation shown above, both cases will path the check, and `ample.eth.` will be considered appropriate to sign for `example.eth.`. This is against the original design of DNS, and would result in breach of zone hierarchy.

In practice, the requirement to exploit this is a bit more complex. Since names are stored as a sequence of packed labels, `example.eth.` should be stored as `\x06example\x03eth\x00`, while `xample.eth.` is stored as `\x05xample\x03eth\x00`. Thus to successfully pull off the attack ,we have to make sure that the packed signer's name is actually a substring of child zone.

A simple (yet unrealistic) example can be like this `xample.eth.` can sign for `e\x05xample.eth.`, since packed format of those two names are `\x05xample\x03eth\x00` and `\x07e\x05ample\x03eth\x00`.

In general, it would require some effort for an attacker to find attackable zones, nevertheless, this should still be considered as a potential threat to the integrity of ENS.

Recommended Mitigation Steps

Check label by label instead of comparing the entire name.
To actually meet all requirements specified in RFC4034 and RFC4035, there are still a lot to do, but we will discuss that in a separate issue for clarity.

Arachnid (ENS) disagreed with severity and commented:

> This is a valid issue, but unexploitable in the wild; RFC 1035 specifies labels are limited to 63 octets or less, and the ASCII code of lowercase 'a' is 97. As a result, no vulnerable names should exist.

> Recommend that this be triaged as severity 2 as a result.

LSDan (judge) decreased severity to Medium and commented:

> I agree with @Arachnid on this. The lack of real world likelihood makes this a Medium severity.

[M-06] `BytesUtils` : compare will not revert when the `offset` and `len` exceeds the bytes lengths

Submitted by zzzitron

BytesUtils.sol#L44-L51

Compare will return false answer without reverting when the inputs are not valid.

Proof of Concept

The `compare` function is used for `compareNames` . The names are supposed to be DNS wire format. If the strings are malformed, it is possible to give out-of-range `offset` , `len` , `otheroffset` , and `otherlen` . When it happens, the `compare` will return some false values, without reverting, since the validity of `offset` and `len` are not checked.

```
51
52 // dnssec-oracle/BytesUtils.sol::compare
53 // The length of self and other are not enforced
54
55  44     function compare(bytes memory self, uint offset, uint
56  45         uint shortest = len;
57  46         if (otherlen < len)
58  47         shortest = otherlen;
59  48
60  49         uint selfptr;
```

```
61   50          uint otherptr;
```

Recommended Mitigation Steps

Check whether the `offset`, `len` are within the length of `self`, as well as for the `other`.

makoto (ENS) acknowledged and commented:

> It's lacking enough test to prove the bug.

Arachnid (ENS) confirmed and commented:

> `compareNames` does not sanity check the lengths passed in, so this is a legitimate bug.

[M-07] If `PARENT_CANNOT_CONTROL` is set on subdomain, it can be unwrapped then wrapped by its owner and then parent can control it again before the expiry

Submitted by panprog

NameWrapper.sol#L955-L961

There is a general incorrect logic of allowing to burn only `PARENT_CANNOT_CONTROL` fuse without burning `CANNOT_UNWRAP` fuse. If only `PARENT_CANNOT_CONTROL` fuse is burnt, then domain can be unwrapped by its owner and then wrapped again, which clears `PARENT_CANNOT_CONTROL` fuse, making it possible for parent to bypass the limitation of parent control before the expiry.

Bypassing parent control scenario:

1. Alice registers and wraps `test.eth` domain
2. Alice creates subdomain `bob.test.eth` and burns `PARENT_CANNOT_CONTROL` fuse with max expiry, transferring this domain to Bob

3. At this point Bob can verify that he is indeed domain owner of `bob.test.eth` in `NameWrapper`, `PARENT_CANNOT_CONTROL` fuse is burnt for this domain and fuse expiry is set to expiry of `test.eth` domain. So Bob thinks his domain is secure and can not be taken from him before the expiry.
4. Bob unwraps `bob.test.eth` domain.
5. Bob wraps `bob.test.eth` domain, which clears fuses and expiry
6. Alice changes `bob.test.eth` domain ownership to her breaking Bob's impression that his domain was secure until expiry.

Proof of Concept

Copy this to test/wrapper and run:
yarn test test/wrapper/NameWrapperBypassPCC.js

https://gist.github.com/panprog/71dea0fd1875b4d7d5849f7da822ea8b

Recommended Mitigation Steps

Burning any fuse (including `PARENT_CANNOT_CONTROL`) must require `CANNOT_UNWRAP` fuse to be burned (because otherwise it's possible to unwrap+wrap to clear that fuse).

In `NameWrapper._canFusesBeBurned`, condition should be different:

```
        if (
            fuses & ~CANNOT_UNWRAP != 0 &&
            fuses & (PARENT_CANNOT_CONTROL | CANNOT_UNWRAP) !=
            (PARENT_CANNOT_CONTROL | CANNOT_UNWRAP)
        ) {
            revert OperationProhibited(node);
        }
```

jefflau (ENS) commented:

> A mitigation step for this could be to not burn fuses when unwrapping domains to prevent the `PARENT_CANNOT_CONTROL` from being reset.

Arachnid (ENS) disagreed with severity and commented:

> Since this has to be self-inflicted by the victim, this should be severity 2.

LSDan (judge) decreased severity to Medium and commented:

> Agree with the sponsor that this is a medium severity issue due to the external requirement that Bob unwraps and rewraps the domain. Additionally, this requires Alice to all of a sudden become a bad actor, making it highly unlikely that this would occur.

[M-08] Wrong Equals Logic

Submitted by 0x1f8b, also found by alan724

BytesUtils.sol#L115-L127

`equals` with offset might return true when `equals` without offset returns false.

Proof of Concept

The problem is that `self.length` could be greater than `other.length + offset`, it should be `==`, or it should contain a length argument.

Here you have an example of the failure:

- `equals(0x0102030000, 0, 0x010203) => return true`

```
decoded input   {
        "bytes self": "0x0102030000",
        "uint256 offset": "0",
        "bytes other": "0x010203"
}
decoded output  {
        "0": "bool: true"
}
```

Recommended Mitigation Steps

```
    function equals(bytes memory self, uint offset, bytes memory
-       return self.length >= offset + other.length && equals(se
+       return self.length == offset + other.length && equals(se
    }
```

makoto (ENS) confirmed

[M-09] The `unwrapETH2LD` use `transferFrom` instead of `safeTransferFrom` to transfer ERC721 token

Submitted by 0x29A, also found by Amithuddar, benbaessler, berndartmueller, cccz, CRYP70, rbserver, RedOneN, and Sm4rty

NameWrapper.sol#L327-L346

The `unwrapETH2LD` use `transferFrom` to transfer ERC721 token, the `newRegistrant` could be an unprepared contract.

Proof of Concept

Should a ERC-721 compatible token be transferred to an unprepared contract, it would end up being locked up there. Moreover, if a contract explicitly wanted to reject ERC-721 safeTransfers.
Plus take a look to the OZ safeTransfer comments:
`Usage of this method is discouraged, use safeTransferFrom whenever possible.`

Recommended Mitigation Steps

```
    function unwrapETH2LD(
        bytes32 labelhash,
        address newRegistrant,
        address newController
    ) public override onlyTokenOwner(_makeNode(ETH_NODE, labelha
```

```
            _unwrap(_makeNode(ETH_NODE, labelhash), newController);
-           registrar.transferFrom(
+           registrar.safeTransferFrom(
                address(this),
                newRegistrant,
                uint256(labelhash)
            );
        }
```

jefflau (ENS) disputed and commented:

> Transfer is to the contract itself, so there is no point in using `safeTransferFrom`. For other situations where `transferFrom` the behaviour is intended.

LSDan (judge) commented:

> Transfer is to the contract itself, so there is no point in using `safeTransferFrom`. For other situations where `transferFrom` the behaviour is intended.

> That's incorrect in the report above. This is transferring from, not to, the contract itself.

jefflau (ENS) commented:

> That's incorrect in the report above. This is transferring from, not to, the contract itself.

> Yes sorry, that is true. I was replying to some of the duplicates that I closed such as: #126, #147.

[M-10] Incorrect implementation of `RRUtils.serialNumberGte`

Submitted by GimelSec, also found by Lambda and zzzitron

RRUtils.sol#L266-L268

Comparing serial numbers should follow RFC1982 due to the possibility of numbers wrapping around. `RRUtils.serialNumberGte` tried to follow the RFC but failed to do so, leading to incorrect results in comparison.

Proof of Concept

For a serial number i1 to be greater than i2, the rules provided by RFC1982 is as follow

```
((i1 < i2) && ((i2 - i1) > (2**31))) || ((i1 > i2) && ((i1 - i2) < (2**31)))
```

ENS implements `int32(i1) - int32(i2) > 0`, which will suffer from revert in cases such as `i1=0x80000000, i2=0x7fffffff`

Recommended Mitigation Steps

Use the naive implementation instead

```
return (i1 == i2) || ((i1 < i2) && ((i2 - i1) > (2**31))) || ((i1 > i2) && ((i1 - i2) < (2**31)));
```

Arachnid (ENS) disagreed with severity and commented:

> This is intentional, see https://en.wikipedia.org/wiki/Serial_number_arithmetic#General_solution

> Nevertheless, this should be documented. Recommend triaging as QA.

Arachnid (ENS) commented:

> Correction - while the operation is correct per the Wikipedia article, it should be in an `unchecked` block to allow overflow. Recommend triaging as 2.

LSDan (judge) decreased severity to Medium and commented:

> Agree with the sponsor. This is a bug and it does potentially impact protocol functionality, but it will not occur until far in the future, making it fairly unlikely. Medium makes sense here.

[M-11] The preimage DB (i.e., `NameWrapper.names`) can be maliciously manipulated/corrupted

Submitted by PwnedNoMore

[NameWrapper.sol#L520]

By design, the `NameWrapper.names` is used as a preimage DB so that the client can query the domain name by providing the token ID. The name should be correctly stored. To do so, the `NameWrapper` record the domain's name every time it gets wrapped. And as long as all the parent nodes are recorded in the DB, wrapping a child node will be very efficient by simply querying the parent node's name.

However, within a malicious scenario, it is possible that a subdomain can be wrapped without recording its info in the preimage DB.

Specifically, when `NameWrappper.setSubnodeOwner` / `NameWrappper.setSubnodeRecord` on a given subdomain, the following code is used to check whether the subdomain is wrapped or not. The preimage DB is only updated when the subdomain is not wrapped (to save gas I beieve).

```
    function setSubnodeOwner(
        bytes32 parentNode,
        string calldata label,
        address newOwner,
        uint32 fuses,
        uint64 expiry
    )
        public
        onlyTokenOwner(parentNode)
        canCallSetSubnodeOwner(parentNode, keccak256(bytes(label)))
        returns (bytes32 node)
    {
        bytes32 labelhash = keccak256(bytes(label));
        node = _makeNode(parentNode, labelhash);
        (, , expiry) = _getDataAndNormaliseExpiry(parentNode, node,
        if (ens.owner(node) != address(this)) {
            ens.setSubnodeOwner(parentNode, labelhash, address(this)
            _addLabelAndWrap(parentNode, node, label, newOwner, fuse
        } else {
```

```
            _transferAndBurnFuses(node, newOwner, fuses, expiry);
        }
    }
```

However, the problem is that `ens.owner(node) != address(this)` is not sufficient to check whether the node is alreay wrapped. The hacker can manipulate this check by simply invoking `EnsRegistry.setSubnodeOwner` to set the owner as the `NameWrapper` contract without wrapping the node.

Consider the following attack scenario.

- the hacker registers a 2LD domain, e.g., `base.eth`
- he assigns a subdomain for himself, e.g., `sub1.base.eth`
    - the expiry of `sub1.base.eth` should be set as expired shortly
    - note that the expiry is for `sub1.base.eth` instead of `base.eth`, so it is safe to make it soonly expired
- the hacker waits for expiration and unwraps his `sub1.base.eth`
- the hacker invokes `ens.setSubnodeOwner` to set the owner of `sub2.sub1.base.eth` as NameWrapper contract
- the hacker re-wraps his `sub1.base.eth`
- the hacker invokes `nameWrapper.setSubnodeOwner` for `sub2.sub1.base.eth`
    - as such, `names[namehash(sub2.sub1.base.eth)]` becomes empty
- the hacker invokes `nameWrapper.setSubnodeOwner` for `eth.sub2.sub1.base.eth`.
    - as such, `names[namehash(eth.sub2.sub1.base.eth)]` becomes `\x03eth`

It is not rated as a High issue since the forged name is not valid, i.e., without the tailed `\x00` (note that a valid name should be like `\x03eth\x00`). However, the preimage BD can still be corrupted due to this issue.

Suggested Fix

When wrapping node `X`, check whether `NameWrapper.names[X]` is empty directly, and update the preimage DB if it is empty.

Proof of Concept / Attack Scenario

For full details, please see original warden submission.

jefflau (ENS) confirmed

[M-12] `ERC1155Fuse: _transfer` does not revert when sent to the old owner

Submitted by zzzitron

The `safeTransferFrom` does not comply with the ERC1155 standard when the token is sent to the old owner.

Proof of Concept

According to the EIP-1155 standard for the `safeTransferFrom`:

> MUST revert if balance of holder for token `_id` is lower than the `_value` sent.

Let's say `alice` does not hold any token of `tokenId`, and `bob` holds one token of `tokenId`. Then alice tries to send one token of `tokenId` to bob with `safeTranferFrom(alice, bob, tokenId, 1, "")`. In this case, even though alice's balance (= 0) is lower than the amount (= 1) sent, the `safeTransferFrom` will not revert. Thus, violating the EIP-1155 standard.
It can cause problems for other contracts using this token, since they assume the token was transferred if the `safeTransferFrom` does not revert. However, in the example above, no token was actually transferred.

```
284
285 // wrapper/ERC1155Fuse.sol::_transfer
286 // ERC1155Fuse::safeTransferFrom uses _transfer
```

```
287
288 274     function _transfer(
289 275         address from,
290 276         address to,
291 277         uint256 id,
292 278         uint256 amount,
293 279         bytes memory data
294 280     ) internal {
295 281         (address oldOwner, uint32 fuses, uint64 expiry) =
296 282         if (oldOwner == to) {
297 283             return;
298 284         }
```

Recommended Mitigation Steps

Revert even if the `to` address already owns the token.

jefflau (ENS) confirmed and commented:

> Recommend severity QA.

LSDan (judge) commented:

> I'm going to leave this as Medium. This issue could definitely impact other protocols and potentially cause a loss of funds given external factors.

[M-13] Users can create extra ENS records at no cost

Submitted by wastewa, also found by bin2chen, Limbooo, PwnedNoMore, and ronnyx2017

ETHRegistrarController.sol#L249-L268
ETHRegistrarController.sol#L125
BaseRegistrarImplementation.sol#L106

Users using the `register` function in `ETHRegistrarController.sol`, can create an additional bogus ENS entry (Keep the ERC721 and all the glory for as long as they want) for free by exploiting the `functionCall` in the `_setRecords` function.
The only check there (in the setRecord function) is that the nodehash matches the

originally registered ENS entry, this is extremely dangerous because the rest of the functionCall is not checked and the controller has very elevated privileges in ENS ecosystem (and probably beyond).

The single exploit I am showing is already very bad, but I expect there will be more if this is left in. An example of a potential hack is that some of the functions in other ENS contracts (which give the RegistrarController elevated privilege) have dynamic types as the first variables—if users can generate a hash that is a low enough number, they will be able to unlock more exploits in the ENS ecosystem because of how dynamic types are abi encoded. Other developers will probably also trust the `ETHRegistrarController.sol`, so other unknown dangers may come down the road.

The exploit I made (full code in PoC) can mint another ENS entry and keep it for as long as it wants, without paying more—will show code below.

Proof of Concept

Put this code in the `TestEthRegistrarController.js` test suite to run. I just appended this to tests at the bottom of file.

I called the `BaseRegistrarImplementation.register` function with the privileges of `ETHRegistrarController` by passing the base registrar's address as the `resolver` param in the `ETHRegistrarController.register` function call. I was able to set a custom duration at no additional cost.

The final checks of the PoC show that we own two new ENS entries from a single `ETHRegistrarController.register` call. The labelhash of the new bogus ENS entry is the nodehash of the first registered ENS entry.

```
    it('Should allow us to make bogus erc721 token in ENS contract
      const label = 'newconfigname'
      const name = `${label}.eth`
      const node = namehash.hash(name)
      const secondTokenDuration = 788400000 // keep bogus NFT for

      var commitment = await controller.makeCommitment(
        label,
        registrantAccount,
```

```
    REGISTRATION_TIME,
    secret,
    baseRegistrar.address,
    [
      baseRegistrar.interface.encodeFunctionData('register(uir
        node,
        registrantAccount,
        secondTokenDuration
      ]),
    ],
    false,
    0,
    0
  )
  var tx = await controller.commit(commitment)
  expect(await controller.commitments(commitment)).to.equal(
    (await web3.eth.getBlock(tx.blockNumber)).timestamp
  )

  await evm.advanceTime((await controller.minCommitmentAge()).
  var balanceBefore = await web3.eth.getBalance(controller.adc

  let tx2 = await controller.register(
    label,
    registrantAccount,
    REGISTRATION_TIME,
    secret,
    baseRegistrar.address,
    [
      baseRegistrar.interface.encodeFunctionData('register(uir
        node,
        registrantAccount,
        secondTokenDuration
      ]),
    ],
    false,
    0,
    0,
    { value: BUFFERED_REGISTRATION_COST }
  )

  expect(await nameWrapper.ownerOf(node)).to.equal(registrantA
  expect(await ens.owner(namehash.hash(name))).to.equal(nameWr


  expect(await baseRegistrar.ownerOf(node)).to.equal( // this
```

```
        registrantAccount
      )
      expect(await baseRegistrar.ownerOf(sha3(label))).to.equal(
        nameWrapper.address
      )
    })
```

Tools Used

chai tests in repo

Recommended Mitigation Steps

I recommend being stricter on the signatures of the user-provided `resolver` and the function that is being called (like safeTransfer calls in existing token contracts). An example of how to do this is by creating an interface that ENS can publish for users that want to compose their own resolvers and call that instead of a loose functionCall. Users will be free to handle data however they like, while restricting the space of things that can go wrong.

I will provide a loose example here:

```
interface IUserResolver {
    function registerRecords(bytes32 nodeId, bytes32 labelHash,

}
```

Arachnid (ENS) confirmed

jefflau (ENS) disagreed with severity and commented:

> We left this as high severity, but this is a duplicate of this: **#222 comment**.

> I believe this still a low severity, or at a minimum medium.

> The only thing you can pass to `register` is the node, as the `require` inside `setRecords` checks the nodehash. However `register` in the baseRegistrar itself takes a `label` not the namehash of the name, so it will register a name with the

> hash of `namehash(namehash('eth') + node)` , which will be a very useless name as the label will then be a 32 byte keccak hash so `0x123...abc.eth` .

> In the warden's test he tests for the node of the account they originally wanted to buy, not the bogus nft:

```
expect(await baseRegistrar.ownerOf(node)).to.equal( // this chec
      registrantAccount
    )
```

> To test for the bogus nft they would need to do:

```
const node2 = sha3(namehash('eth') + node)
expect(await baseRegistrar.ownerOf(node2)).to.equal( // this che
      registrantAccount
    )
```

LSDan (judge) decreased severity to Medium and commented:

> After a lot of consideration of this issue, I'm going to downgrade it to medium. There are two main considerations in doing this:

1. The "free" name created is essentially junk.
2. The additional potential exploits are unknown and kinda hand-wavy. If there are additional exploits in the future as a result of this they almost definitely rely on external factors that don't exist today.

Low Risk and Non-Critical Issues

For this contest, 71 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by IIIIIII received the top score from the judge.

The following wardens also submitted reports: Dravee, 0x29A, Bnke0x0, Deivitto, joestakey, rbserver, 0x1f8b, benbaessler, TomJ, dxdv, hake, Rolezn, 0xNazgul, 0xf15ers, alan724, Sm4rty, Funen, sashik_eth, Ruhum, robee, _Adam, Aussie_Battlers, Waze, brgltd, c3phas, Ch_301, hyh, Lambda, MiloTruck,

p_crypt0, Rohan16, 0xNineDec, 8olidity, zzzitron, GimelSec, JC, JohnSmith, kyteg, rokinot, asutorufos, berndartmueller, bulej93, cRat1st0s, Critical, csanuragjain, delfin454000, fatherOfBlocks, sach1r0, pedr02b2, philogy, PwnedNoMore, rajatbeladiya, __141345__, 0xDjango, rishabh, zuhaibmohd, cryptphi, svskaushik, seyni, RustyRabbit, lcfr_eth, minhtrng, ReyAdmirado, pashov, bin2chen, cryptonue, ElKu, exd0tpy, simon135, *and* gogo.

Low Risk Issues Summary

	Issue	Instances
[L-01]	`require()` should be used instead of `assert()`	2
[L-02]	Missing checks for `address(0x0)` when assigning values to `address` state variables	1
[L-03]	Open TODOs	1
[L-04]	File is missing NatSpec	10
[L-05]	NatSpec is incomplete	13

Total: 27 instances over 5 issues

Non-critical Issues Summary

	Issue	Instances
[N-01]	Name validation is not strictly valid	1
[N-02]	Adding a `return` statement when the function defines a named return variable, is redundant	1
[N-03]	`require()` / `revert()` statements should have descriptive reason strings	17
[N-04]	`public` functions not called by the contract should be declared `external` instead	13
[N-05]	`constant` s should be defined rather than using magic numbers	150

	Issue	Instances
[N-06]	Redundant cast	1
[N-07]	Missing event and or timelock for critical parameter change	3
[N-08]	File is missing version pragma	1
[N-09]	Use a more recent version of solidity	7
[N-10]	`pragma experimental ABIEncoderV2` is deprecated	2
[N-11]	Constant redefined elsewhere	1
[N-12]	Inconsistent spacing in comments	2
[N-13]	Lines are too long	8
[N-14]	Inconsistent method of specifying a floating pragma	10
[N-15]	Non-library/interface files should use fixed compiler versions, not floating ones	4
[N-16]	Typos	5
[N-17]	File does not contain an SPDX Identifier	14
[N-18]	Event is missing `indexed` fields	18
[N-19]	Not using the named return variables anywhere in the function is confusing	4
[N-20]	Duplicated `require()` / `revert()` checks should be refactored to a modifier or function	6

Total: 268 instances over 20 issues

[L-01] `require()` should be used instead of `assert()`

Prior to solidity version 0.8.0, hitting an assert consumes the **remainder of the transaction's available gas** rather than returning it, as `require()` / `revert()` do.

`assert()` should be avoided even past solidity version 0.8.0 as its documentation states that “The assert function creates an error of type Panic(uint256). ... Properly functioning code should never create a Panic, not even on invalid external input. If this happens, then there is a bug in your contract which you should fix”.

There are 2 instances of this issue. (For in-depth details on this and all further issues with multiple instances, please see the warden’s full report.)

[L-02] Missing checks for `address(0x0)` when assigning values to `address` state variables

There is 1 instance of this issue:

```
File: contracts/dnssec-oracle/Owned.sol

19:             owner = newOwner;
```

https://github.com/code-423n4/2022-07-ens/blob/ff6e59b9415d0ead7daf31c2ed06e86d9061ae22/contracts/dnssec-oracle/Owned.sol#L19

[L-03] Open TODOs

Code architecture, incentives, and error handling/reporting questions/issues should be resolved before deployment

There is 1 instance of this issue:

```
File: contracts/dnssec-oracle/DNSSECImpl.sol

238:             // TODO: Check key isn't expired, unless updating
```

https://github.com/code-423n4/2022-07-ens/blob/ff6e59b9415d0ead7daf31c2ed06e86d9061ae22/contracts/dnssec-oracle/DNSSECImpl.sol#L238

[L-04] File is missing NatSpec

There are 10 instances of this issue.

[L-05] NatSpec is incomplete

There are 13 instances of this issue.

[N-01] Name validation is not strictly valid

While the documentation does in fact say that there are other validations necessary to be compatible with the legacy DNS system, it would be better to have the following function signature instead `function valid(string calldata name, bool isEnforceLegacyRules) public pure returns (bool)` , so it's clear what the caller is validating

There is 1 instance of this issue:

```
File: contracts/ethregistrar/ETHRegistrarController.sol

77        function valid(string memory name) public pure returns
78            return name.strlen() >= 3;
79:       }
```

https://github.com/code-423n4/2022-07-ens/blob/4dfb0e32f586bff3db486349523a93480e3ddfba/contracts/ethregistrar/ETHRegistrarController.sol#L77-L79

[N-02] Adding a `return` statement when the function defines a named return variable, is redundant

There is 1 instance of this issue:

```
File: contracts/dnssec-oracle/DNSSECImpl.sol

139:          return rrset;
```

https://github.com/code-423n4/2022-07-ens/blob/ff6e59b9415d0ead7daf31c2ed06e86d9061ae22/contracts/dnssec-oracle/DNSSECImpl.sol#L139

[N-03] `require()` / `revert()` statements should have descriptive reason strings

There are 17 instances of this issue.

[N-04] `public` functions not called by the contract should be declared `external` instead

Contracts are allowed to override their parents' functions and change the visibility from `external` to `public`.

There are 13 instances of this issue.

[N-05] `constant`s should be defined rather than using magic numbers

Even assembly can benefit from using readable constants instead of hex/numeric literals.

There are 150 instances of this issue.

[N-06] Redundant cast

The type of the variable is the same as the type to which the variable is being cast.

There is 1 instance of this issue:

```
File: contracts/registry/ReverseRegistrar.sol

/// @audit address(resolver)
53:                 address(resolver) != address(0),
```

https://github.com/code-423n4/2022-07-ens/blob/ff6e59b9415d0ead7daf31c2ed06e86d9061ae22/contracts/registry/ReverseRegistrar.sol#L53

[N-07] Missing event and or timelock for critical parameter change

Events help non-contract tools to track changes, and events prevent users from being surprised by changes

There are 3 instances of this issue.

[N-08] File is missing version pragma

There is 1 instance of this issue:

```
File: contracts/ethregistrar/IBaseRegistrar.sol

0:    import "../registry/ENS.sol";
```

https://github.com/code-423n4/2022-07-ens/blob/ff6e59b9415d0ead7daf31c2ed06e86d9061ae22/contracts/ethregistrar/IBaseRegistrar.sol#L0

[N-09] Use a more recent version of solidity

- Use a solidity version of at least 0.8.12 to get `string.concat()` to be used instead of `abi.encodePacked(<str>,<str>)`
  There are 4 instances of this issue.
- Use a solidity version of at least 0.8.13 to get the ability to use `using for` with a list of free functions
  There are 3 instances of this issue.

[N-10] `pragma experimental ABIEncoderV2` is deprecated

Use `pragma abicoder v2` instead

There are 2 instances of this issue.

[N-11] 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.

There is 1 instance of this issue:

```
File: contracts/wrapper/NameWrapper.sol

/// @audit seen in contracts/registry/ReverseRegistrar.sol
35:       ENS public immutable override ens;
```

https://github.com/code-423n4/2022-07-ens/blob/ff6e59b9415d0ead7daf31c2ed06e86d9061ae22/contracts/wrapper/NameWrapper.sol#L35

[N-12] Inconsistent spacing in comments

Some lines use `// x` and some use `//x`. The instances below point out the usages that don't follow the majority, within each file

There are 2 instances of this issue.

[N-13] Lines are too long

Usually lines in source code are limited to 80 characters. Today's screens are much larger so it's reasonable to stretch this in some cases. Since the files will most likely reside in GitHub, and GitHub starts using a scroll bar in all cases when the length is over 164 characters, the lines below should be split when they reach that length

There are 8 instances of this issue.

[N-14] 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

There are 10 instances of this issue.

[N-15] Non-library/interface files should use fixed compiler versions, not floating ones

There are 4 instances of this issue.

[N-16] Typos

There are 5 instances of this issue.

[N-17] File does not contain an SPDX Identifier

There are 14 instances of this issue.

[N-18] 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.

There are 18 instances of this issue.

[N-19] Not using the named return variables anywhere in the function is confusing

Consider changing the variable to be an unnamed one.

There are 4 instances of this issue.

[N-20] Duplicated `require()` / `revert()` checks should be refactored to a modifier or function

The compiler will inline the function, which will avoid `JUMP` instructions usually associated with functions

There are 6 instances of this issue.

jefflau (ENS) commented:

> High quality submission, documented well with links and code examples.

Gas Optimizations

For this contest, 70 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **0xKitsune** received the top score from the judge.

The following wardens also submitted reports: joestakey, lllllll, gogo, Dravee, m_Rassska, MiloTruck, rbserver, hake, TomJ, 0x1f8b, ajtra, Bnke0x0, c3phas, Deivitto, kyteg, RedOneN, Sm4rty, Tomio, __141345__, brgltd, bulej93, Ch_301, cRat1st0s, durianSausage, JC, JohnSmith, _Adam, 0xNazgul, asutorufos, delfin454000, fatherOfBlocks, rokinot, Waze, 0xNineDec, 0x040, 0x29A, 0xsam, Aussie_Battlers, Aymen0909, Fitraldys, lucacez, Noah3o6, ReyAdmirado, robee, Rohan16, Rolezn, sach1r0, samruna, 8olidity, arcoun, benbaessler, CRYP70, karanctf, lcfr_eth, rajatbeladiya, sahar, sashik_eth, simon135, zuhaibmohd, cryptonue, ElKu, Funen, hyh, seyni, ak1, Chom, GimelSec, Lambda, *and* Ruhum.

Gas Optimizations Summary

The following sections detail the gas optimizations found throughout the codebase. Each optimization is documented with the setup, an explainer for the optimization, a gas report and line identifiers for each optimization across the codebase. For each section's gas report, the optimizer was turned on and set to 10000 runs. You can replicate any tests/gas reports by heading to 0xKitsune/gas-lab and cloning the

repo. Then, simply copy/paste the contract examples from any section and run `forge test --gas-report`. You can also easily update the optimizer runs in the `foundry.toml`.

[G-01] Use assembly to hash instead of Solidity

```
contract GasTest is DSTest {
    Contract0 c0;
    Contract1 c1;

    function setUp() public {
        c0 = new Contract0();
        c1 = new Contract1();
    }

    function testGas() public view {
        c0.solidityHash(2309349, 2304923409);
        c1.assemblyHash(2309349, 2304923409);
    }
}

contract Contract0 {
    function solidityHash(uint256 a, uint256 b) public view {
        //unoptimized
        keccak256(abi.encodePacked(a, b));
    }
}

contract Contract1 {
    function assemblyHash(uint256 a, uint256 b) public view {
        //optimized
        assembly {
            mstore(0x00, a)
            mstore(0x20, b)
            let hashedVal := keccak256(0x00, 0x40)
        }
    }
}
```

Gas Report

Contract0 contract					
Deployment Cost	Deployment Size				
36687	214				
Function Name	min	avg	median	max	
solidityHash	313	313	313	313	

Contract1 contract					
Deployment Cost	Deployment Size				
31281	186				
Function Name	min	avg	median	max	
assemblyHash	231	231	231	231	

Lines

For full list of line references, see warden's original submission.

[G-02] `unchecked{++i}` instead of `i++` (or use assembly when applicable)

Use `++i` instead of `i++`. This is especially useful in for loops but this optimization can be used anywhere in your code. You can also use `unchecked{++i;}` for even more gas savings but this will not check to see if `i` overflows. For extra safety if you are worried about this, you can add a require statement after the loop checking if `i` is equal to the final incremented value. For best gas savings, use inline assembly, however this limits the functionality you can achieve. For example you cant use Solidity syntax to internally call your own contract within an assembly block and external calls must be done with the `call()` or `delegatecall()` instruction. However when applicable, inline assembly will save much more gas.

```
contract GasTest is DSTest {
    Contract0 c0;
    Contract1 c1;
    Contract2 c2;
    Contract3 c3;
    Contract4 c4;

    function setUp() public {
        c0 = new Contract0();
        c1 = new Contract1();
        c2 = new Contract2();
        c3 = new Contract3();
        c4 = new Contract4();
    }

    function testGas() public {
        c0.iPlusPlus();
        c1.plusPlusI();
        c2.uncheckedPlusPlusI();
        c3.safeUncheckedPlusPlusI();
        c4.inlineAssemblyLoop();
    }
}

contract Contract0 {
    //loop with i++
    function iPlusPlus() public pure {
        uint256 j = 0;
        for (uint256 i; i < 10; i++) {
            j++;
        }
    }
}

contract Contract1 {
    //loop with ++i
    function plusPlusI() public pure {
        uint256 j = 0;
        for (uint256 i; i < 10; ++i) {
            j++;
        }
    }
}

contract Contract2 {
```

```
    //loop with unchecked{++i}
    function uncheckedPlusPlusI() public pure {
        uint256 j = 0;
        for (uint256 i; i < 10; ) {
            j++;

            unchecked {
                ++i;
            }
        }
    }
}

contract Contract3 {
    //loop with unchecked{++i} with additional overflow check
    function safeUncheckedPlusPlusI() public pure {
        uint256 j = 0;
        uint256 i = 0;
        for (i; i < 10; ) {
            j++;

            unchecked {
                ++i;
            }
        }

        //check for overflow
        assembly {
            if lt(i, 10) {
                mstore(0x00, "loop overflow")
                revert(0x00, 0x20)
            }
        }
    }
}

contract Contract4 {
    //loop with inline assembly
    function inlineAssemblyLoop() public pure {
        assembly {
            let j := 0

            for {
                let i := 0
            } lt(i, 10) {
                i := add(i, 0x01)
```

```
                } {
                    j := add(j, 0x01)
                }
            }
        }
    }
```

Gas Report

Contract0 contract				
Deployment Cost	Deployment Size			
37687	219			
Function Name	min	avg	median	max
iPlusPlus	2039	2039	2039	2039

Contract1 contract				
Deployment Cost	Deployment Size			
37287	217			
Function Name	min	avg	median	max
plusPlusI	1989	1989	1989	1989

Contract3 contract				
Deployment Cost	Deployment Size			
42693	244			
Function Name	min	avg	median	
safeUncheckedPlusPlusI	1355	1355	1355	

Contract2 contract				
Deployment Cost	Deployment Size			
35887	210			
Function Name	min	avg	median	max
uncheckedPlusPlusI	1329	1329	1329	1329

Contract4 contract				
Deployment Cost	Deployment Size			
26881	164			
Function Name	min	avg	median	max
inlineAssemblyLoop	709	709	709	709

Lines

- DNSSECImpl.sol:93
- ETHRegistrarController.sol:256
- BytesUtils.sol:266
- BytesUtils.sol:313
- ERC1155Fuse.sol:92
- ERC1155Fuse.sol:205
- StringUtils.sol:14

[G-03] Use assembly for math (add, sub, mul, div)

Use assembly for math instead of Solidity. You can check for overflow/underflow in assembly to ensure safety. If using Solidity versions < 0.8.0 and you are using Safemath, you can gain significant gas savings by using assembly to calculate values and checking for overflow/underflow.

```
contract GasTest is DSTest {
    Contract0 c0;
    Contract1 c1;
    Contract2 c2;
    Contract3 c3;
    Contract4 c4;
    Contract5 c5;
    Contract6 c6;
    Contract7 c7;

    function setUp() public {
        c0 = new Contract0();
        c1 = new Contract1();
        c2 = new Contract2();
        c3 = new Contract3();
        c4 = new Contract4();
        c5 = new Contract5();
        c6 = new Contract6();
        c7 = new Contract7();
    }

    function testGas() public {
        c0.addTest(34598345, 100);
        c1.addAssemblyTest(34598345, 100);
        c2.subTest(34598345, 100);
        c3.subAssemblyTest(34598345, 100);
        c4.mulTest(34598345, 100);
        c5.mulAssemblyTest(34598345, 100);
        c6.divTest(34598345, 100);
        c7.divAssemblyTest(34598345, 100);
    }
}

contract Contract0 {
    //addition in Solidity
    function addTest(uint256 a, uint256 b) public pure {
        uint256 c = a + b;
    }
}

contract Contract1 {
    //addition in assembly
    function addAssemblyTest(uint256 a, uint256 b) public pure {
        assembly {
            let c := add(a, b)
```

```
            if lt(c, a) {
                mstore(0x00, "overflow")
                revert(0x00, 0x20)
            }
        }
    }
}

contract Contract2 {
    //subtraction in Solidity
    function subTest(uint256 a, uint256 b) public pure {
        uint256 c = a - b;
    }
}

contract Contract3 {
    //subtraction in assembly
    function subAssemblyTest(uint256 a, uint256 b) public pure {
        assembly {
            let c := sub(a, b)

            if gt(c, a) {
                mstore(0x00, "underflow")
                revert(0x00, 0x20)
            }
        }
    }
}

contract Contract4 {
    //multiplication in Solidity
    function mulTest(uint256 a, uint256 b) public pure {
        uint256 c = a * b;
    }
}

contract Contract5 {
    //multiplication in assembly
    function mulAssemblyTest(uint256 a, uint256 b) public pure {
        assembly {
            let c := mul(a, b)

            if lt(c, a) {
                mstore(0x00, "overflow")
                revert(0x00, 0x20)
```

```
            }
        }
    }
}

contract Contract6 {
    //division in Solidity
    function divTest(uint256 a, uint256 b) public pure {
        uint256 c = a * b;
    }
}

contract Contract7 {
    //division in assembly
    function divAssemblyTest(uint256 a, uint256 b) public pure {
        assembly {
            let c := div(a, b)

            if gt(c, a) {
                mstore(0x00, "underflow")
                revert(0x00, 0x20)
            }
        }
    }
}
```

Gas Report

Contract0 contract				
Deployment Cost	Deployment Size			
40493	233			
Function Name	min	avg	median	max
addTest	303	303	303	303

Contract1 contract				
Deployment Cost	Deployment Size			

37087	216			
Function Name	min	avg	median	max
addAssemblyTest	263	263	263	263

Contract2 contract				
Deployment Cost	Deployment Size			
40293	232			
Function Name	min	avg	median	max
subTest	300	300	300	300

Contract3 contract				
Deployment Cost	Deployment Size			
37287	217			
Function Name	min	avg	median	max
subAssemblyTest	263	263	263	263

Contract4 contract				
Deployment Cost	Deployment Size			
41893	240			
Function Name	min	avg	median	max
mulTest	325	325	325	325

Contract5 contract				
Deployment Cost	Deployment Size			
37087	216			

Function Name	min	avg	median	max
mulAssemblyTest	265	265	265	265

Contract6 contract				
Deployment Cost	Deployment Size			
41893	240			
Function Name	min	avg	median	max
divTest	325	325	325	325

Contract7 contract				
Deployment Cost	Deployment Size			
37287	217			
Function Name	min	avg	median	max
divAssemblyTest	265	265	265	265

Lines

For full list of line references, see warden's original submission.

[G-04] Use `calldata` instead of `memory` for function arguments that do not get mutated.

Mark data types as `calldata` instead of `memory` where possible. This makes it so that the data is not automatically loaded into memory. If the data passed into the function does not need to be changed (like updating values in an array), it can be passed in as `calldata`. The one exception to this is if the argument must later be passed into another function that takes an argument that specifies `memory` storage.

```
contract GasTest is DSTest {
    Contract0 c0;
    Contract1 c1;
    Contract2 c2;
    Contract3 c3;

    function setUp() public {
        c0 = new Contract0();
        c1 = new Contract1();
        c2 = new Contract2();
        c3 = new Contract3();
    }

    function testGas() public {
        uint256[] memory arr = new uint256[](10);
        c0.calldataArray(arr);
        c1.memoryArray(arr);

        bytes memory data = abi.encode("someText");
        c2.calldataBytes(data);
        c3.memoryBytes(data);
    }
}

contract Contract0 {
    function calldataArray(uint256[] calldata arr) public {
        uint256 j;
        for (uint256 i; i < arr.length; i++) {
            j = arr[i] + 10;
        }
    }
}

contract Contract1 {
    function memoryArray(uint256[] memory arr) public {
        uint256 j;
        for (uint256 i; i < arr.length; i++) {
            j = arr[i] + 10;
        }
    }
}

contract Contract2 {
    function calldataBytes(bytes calldata data) public {
        bytes32 val;
```

```
        for (uint256 i; i < 10; i++) {
            val = keccak256(abi.encode(data, i));
        }
    }
}

contract Contract3 {
    function memoryBytes(bytes memory data) public {
        bytes32 val;
        for (uint256 i; i < 10; i++) {
            val = keccak256(abi.encode(data, i));
        }
    }
}
```

Gas Report

src/test/GasTest.t.sol:Contract0 contract	
Deployment Cost	Deployment Size
97947	521
Function Name	min
calldataArray	2824

src/test/GasTest.t.sol:Contract1 contract	
Deployment Cost	Deployment Size
128171	672
Function Name	min
memoryArray	3755

src/test/GasTest.t.sol:Contract2 contract	
Deployment Cost	Deployment Size

```
|-----------------------------------------------------------------|
| 100547                                          | 534           |
|-----------------------------------------------------------------|
| Function Name                                   | min           |
|-----------------------------------------------------------------|
| calldataBytes                                   | 4934          |
|-------------------------------------------------|---------------|

|-------------------------------------------------|---------------|
| src/test/GasTest.t.sol:Contract3 contract       |               |
|=================================================|===============|
| Deployment Cost                                 | Deployment Size |
|-----------------------------------------------------------------|
| 135183                                          | 707           |
|-----------------------------------------------------------------|
| Function Name                                   | min           |
|-----------------------------------------------------------------|
| memoryBytes                                     | 7551          |
|-------------------------------------------------|---------------|
```

Lines

For full list of line references, see warden's [original submission](#).

[G-05] Use multiple require() statements instead of require(expression && expression && ...)

You can save gas by breaking up a require statement with multiple conditions, into multiple require statements with a single condition.

```
contract GasTest is DSTest {
    Contract0 c0;
    Contract1 c1;

    function setUp() public {
        c0 = new Contract0();
        c1 = new Contract1();
    }

    function testGas() public {
        c0.singleRequire(3);
        c1.multipleRequire(3);
    }
}
```

```
contract Contract0 {
    function singleRequire(uint256 num) public {
        require(num > 1 && num < 10 && num == 3);
    }
}

contract Contract1 {
    function multipleRequire(uint256 num) public {
        require(num > 1);
        require(num < 10);
        require(num == 3);
    }
}
```

Gas Report

Contract0 contract				
Deployment Cost	Deployment Size			
35487	208			
Function Name	min	avg	median	max
singleRequire	286	286	286	286

Contract1 contract				
Deployment Cost	Deployment Size			
35887	210			
Function Name	min	avg	median	max
multipleRequire	270	270	270	270

Lines

- BytesUtils.sol:268
- ERC1155Fuse.sol:216
- ERC1155Fuse.sol:291

[G-06] Use assembly to write storage values

```
contract GasTest is DSTest {
    Contract0 c0;
    Contract1 c1;

    function setUp() public {
        c0 = new Contract0();
        c1 = new Contract1();
    }

    function testGas() public {
        c0.updateOwner(0x158B28A1b1CB1BE12C6bD8f5a646a0e3B20247
        c1.assemblyUpdateOwner(0x158B28A1b1CB1BE12C6bD8f5a646a0e
    }
}

contract Contract0 {
    address owner = 0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84;

    function updateOwner(address newOwner) public {
        owner = newOwner;
    }
}

contract Contract1 {
    address owner = 0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84;

    function assemblyUpdateOwner(address newOwner) public {
        assembly {
            sstore(owner.slot, newOwner)
        }
    }
}
```

Gas Report

Contract0 contract				
Deployment Cost	Deployment Size			
60623	261			
Function Name	min	avg	median	max
updateOwner	5302	5302	5302	5302

Contract1 contract				
Deployment Cost	Deployment Size			
54823	232			
Function Name	min	avg	median	max
assemblyUpdateOwner	5236	5236	5236	5236

Lines

- Owned.sol:15
- Owned.sol:19
- ETHRegistrarController.sol:61
- ETHRegistrarController.sol:62

[G-07] Use assembly to check for address(0)

```
contract GasTest is DSTest {
    Contract0 c0;
    Contract1 c1;

    function setUp() public {
        c0 = new Contract0();
        c1 = new Contract1();
    }
```

```
    function testGas() public view {
        c0.ownerNotZero(address(this));
        c1.assemblyOwnerNotZero(address(this));
    }
}

contract Contract0 {
    function ownerNotZero(address _addr) public pure {
        require(_addr != address(0), "zero address)");
    }
}

contract Contract1 {
    function assemblyOwnerNotZero(address _addr) public pure {
        assembly {
            if iszero(_addr) {
                mstore(0x00, "zero address")
                revert(0x00, 0x20)
            }
        }
    }
}
```

Gas Report

Contract0 contract				
Deployment Cost	Deployment Size			
61311	338			
Function Name	min	avg	median	max
ownerNotZero	258	258	258	258

Contract1 contract				
Deployment Cost	Deployment Size			
44893	255			

```
├-------------------------------------┼-------------------------
| Function Name         | min         | avg | median | max
├-------------------------------------┼-------------------------
| assemblyOwnerNotZero  | 252         | 252 | 252    | 252
└-------------------------------------┴-------------------------
```

Lines

- ReverseRegistrar.sol:53
- DNSSECImpl.sol:336
- ETHRegistrarController.sol:100
- NameWrapper.sol:132
- NameWrapper.sol:139
- NameWrapper.sol:318
- NameWrapper.sol:661
- NameWrapper.sol:763
- NameWrapper.sol:799
- NameWrapper.sol:911
- ERC1155Fuse.sol:61
- ERC1155Fuse.sol:176
- ERC1155Fuse.sol:199
- ERC1155Fuse.sol:248
- ERC1155Fuse.sol:249

[G-08] Use assembly when getting a contract's balance of ETH.

You can use `selfbalance()` instead of `address(this).balance` when getting your contract's balance of ETH to save gas. Additionally, you can use `balance(address)` instead of `address.balance()` when getting an external contract's balance of ETH.

```
contract GasTest is DSTest {
    Contract0 c0;
```

```
    Contract1 c1;
    Contract2 c2;
    Contract3 c3;

    function setUp() public {
        c0 = new Contract0();
        c1 = new Contract1();
        c2 = new Contract2();
        c3 = new Contract3();
    }

    function testGas() public {
        c0.addressInternalBalance();
        c1.assemblyInternalBalance();
        c2.addressExternalBalance(address(this));
        c3.assemblyExternalBalance(address(this));
    }
}

contract Contract0 {
    function addressInternalBalance() public returns (uint256) {
        return address(this).balance;
    }
}

contract Contract1 {
    function assemblyInternalBalance() public returns (uint256)
        assembly {
            let c := selfbalance()
            mstore(0x00, c)
            return(0x00, 0x20)
        }
    }
}

contract Contract2 {
    function addressExternalBalance(address addr) public {
        uint256 bal = address(addr).balance;
        bal++;
    }
}

contract Contract3 {
    function assemblyExternalBalance(address addr) public {
        uint256 bal;
        assembly {
```

```
                bal := balance(addr)
            }
            bal++;
        }
    }
```

Gas Report

Contract0 contract				
Deployment Cost	Deployment Size			
23675	147			
Function Name	min	avg	median	m
addressInternalBalance	148	148	148	1

Contract1 contract				
Deployment Cost	Deployment Size			
27081	165			
Function Name	min	avg	median	
assemblyInternalBalance	133	133	133	

Contract2 contract				
Deployment Cost	Deployment Size			
61511	339			
Function Name	min	avg	median	m
addressExternalBalance	417	417	417	4

Contract3 contract				

Deployment Cost	Deployment Size			
57105	317			
Function Name	min	avg	median	
assemblyExternalBalance	411	411	411	

Lines

- ETHRegistrarController.sol:211

[G-09] Cache array length during for loop definition.

A typical for loop definition may look like: `for (uint256 i; i < arr.length; i++){}`. Instead of using `array.length`, cache the array length before the loop, and use the cached value to safe gas. This will avoid an `MLOAD` every loop for arrays stored in memory and an `SLOAD` for arrays stored in storage. This can have significant gas savings for arrays with a large length, especially if the array is stored in storage.

```
contract GasTest is DSTest {
    Contract0 c0;
    Contract1 c1;
    Contract2 c2;
    Contract3 c3;

    function setUp() public {
        c0 = new Contract0();
        c1 = new Contract1();
        c2 = new Contract2();
        c3 = new Contract3();
    }

    function testGas() public view {
        uint256[] memory arr = new uint256[](10);
        c0.nonCachedMemoryListLength(arr);
        c1.cachedMemoryListLength(arr);
        c2.nonCachedStorageListLength();
        c3.cachedStorageListLength();
```

```
    }
}

contract Contract0 {
    function nonCachedMemoryListLength(uint256[] memory arr) pub
        uint256 j;
        for (uint256 i; i < arr.length; i++) {
            j = arr[i] + 10;
        }
    }
}

contract Contract1 {
    function cachedMemoryListLength(uint256[] memory arr) public
        uint256 j;

        uint256 length = arr.length;
        for (uint256 i; i < length; i++) {
            j = arr[i] + 10;
        }
    }
}

contract Contract2 {
    uint256[] arr = new uint256[](10);

    function nonCachedStorageListLength() public view {
        uint256 j;
        for (uint256 i; i < arr.length; i++) {
            j = arr[i] + 10;
        }
    }
}

contract Contract3 {
    uint256[] arr = new uint256[](10);

    function cachedStorageListLength() public view {
        uint256 j;
        uint256 length = arr.length;

        for (uint256 i; i < length; i++) {
            j = arr[i] + 10;
        }
    }
}
```

Gas Report

src/test/GasTest.t.sol:Contract0 contract	
Deployment Cost	Deployment Size
128171	672
Function Name	min
nonCachedMemoryListLength	3755

src/test/GasTest.t.sol:Contract1 contract	
Deployment Cost	Deployment Size
128777	675
Function Name	min
cachedMemoryListLength	3733

src/test/GasTest.t.sol:Contract2 contract	
Deployment Cost	Deployment Size
118474	539
Function Name	min
nonCachedStorageListLength	27979

src/test/GasTest.t.sol:Contract3 contract	
Deployment Cost	Deployment Size
118674	540

Function Name	min
cachedStorageListLength	26984

Lines

- DNSSECImpl.sol:93
- ETHRegistrarController.sol:256
- ERC1155Fuse.sol:92
- ERC1155Fuse.sol:205
- RRUtils.sol:310

[G-10] Consider marking functions as payable

You can mark public or external functions as payable to save gas. Functions that are not payable have additional logic to check if there was a value sent with a call, however, making a function payable eliminates this check. This optimization should be carefully considered due to potentially unwanted behavior when a function does not need to accept ether.

```
contract GasTest is DSTest {
    Contract0 c0;
    Contract1 c1;

    function setUp() public {
        c0 = new Contract0();
        c1 = new Contract1();
    }

    function testGas() public {
        c0.isNotPayable();
        c1.isPayable();
    }
}

contract Contract0 {
    function isNotPayable() public view {
        uint256 val = 0;
        val++;
```

```
    }
}

contract Contract1 {
    function isPayable() public payable {
        uint256 val = 0;
        val++;
    }
}
```

Gas Report

Contract0 contract				
Deployment Cost	Deployment Size			
32081	190			
Function Name	min	avg	median	max
isNotPayable	198	198	198	198

Contract1 contract				
Deployment Cost	Deployment Size			
29681	178			
Function Name	min	avg	median	max
isPayable	174	174	174	174

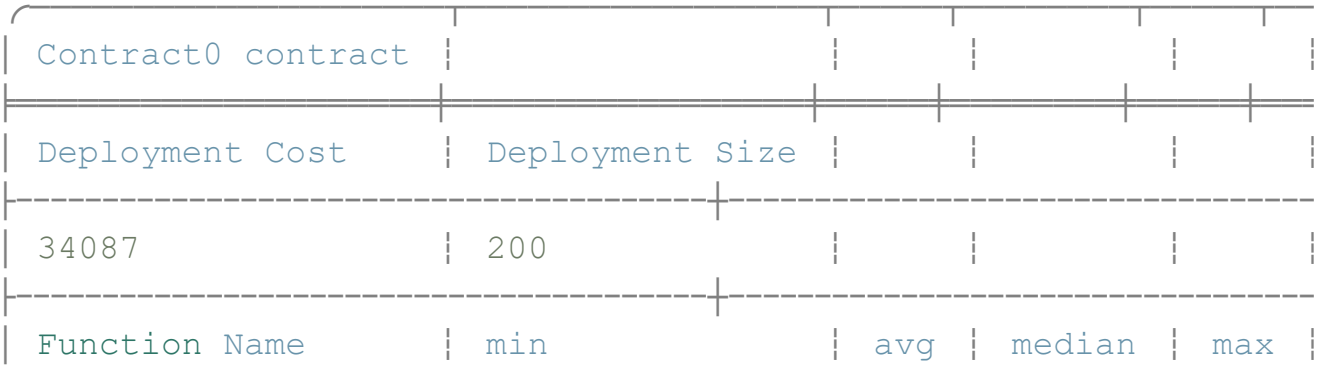
Lines

For full list of line references, see warden's [original submission](#).

[G-11] Use custom errors instead of string error messages

```
contract GasTest is DSTest {
    Contract0 c0;
    Contract1 c1;

    function setUp() public {
        c0 = new Contract0();
        c1 = new Contract1();
    }

    function testFailGas() public {
        c0.stringErrorMessage();
        c1.customErrorMessage();
    }
}

contract Contract0 {
    function stringErrorMessage() public {
        bool check = false;
        require(check, "error message");
    }
}

contract Contract1 {
    error CustomError();

    function customErrorMessage() public {
        bool check = false;
        if (!check) {
            revert CustomError();
        }
    }
}
```

Gas Report



stringErrorMessage	218	218	218	218

Contract1 contract				
Deployment Cost	Deployment Size			
26881	164			
Function Name	min	avg	median	max
customErrorMessage	161	161	161	161

Lines

For full list of line references, see warden's original submission.

jefflau (ENS) commented:

> High quality submission with gas tables and reproduction.

Disclosures

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

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest 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