ENS (Ethereum Name Service)

Scope

The code under review can be found within the C4 ENS repository.

Summary

Findings

ID	Issue	Severity
N-01	Missing event emission for critical state changes	Non-Critical
N-02	Unnecessary typecast to uint256	Non-Critical
N-03	Remove unnecessary code	Non-Critical
N-04	Constants should be defined rather than using magic numbers	Non-Critical
L-01	Missing getUri() function that returns baseUri appended with a user specific tokenId	Low
R-01	Consider adding a time interval field	Recommendation

Gas Optimizations

Gas Optimizations	Issues	Instances
G-01	Unnecessary function can be removed	1
G-02	Use do-while loop instead of for-loop to save gas	1
G-03	Remove initialization to 0 to save gas	1
G-04	Remove parameter `ERC20Votes _token` from function `retrieveProxyContractAddress` to save gas	1
G-05	Use if conditional statements instead of ternary operators to save gas	1
G-06	Use gas-efficient assembly for common math operations like min and max	3
G-07	Consider using alternatives to OpenZeppelin	1
G-08	Remove return statement to save gas	1

Analysis Report

- Comments for the judge to contextualize my findings
- Approach taken in evaluating the codebase
- Architecture Recommendations
    - What's unique?
    - What's using existing patterns and how this codebase compare to others I'm familiar with
    - What ideas can be incorporated?
- Codebase quality analysis
- Centralization risks
- Resources used to gain deeper context on the codebase
- Mechanism Review
- High-level System Overview
- Documentation/Mental models
- User-experience weak spots and how they can be mitigated
- Areas of Concern, Attack surfaces, Invariants - QnA
    - Some questions I asked myself
- Time spent

Findings

[N-01] Missing event emission for critical state changes

There are 4 instances of this issue:

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L15C1-L21C1

Missing event emission when new delegator instance is created.

```
File: ERC20MultiDelegate.sol
16:     constructor(ERC20Votes _token, address _delegate) {
17:         _token.approve(msg.sender, type(uint256).max);
18:         _token.delegate(_delegate);
19:         //@audit NC - missing event emission when new delegator
instance is created
20:     }
```

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L44C1-L48C7

Missing event emission when `token` and `uri` are set. Note: ERC1155 does not emit an event when setting a uri as seen here.

```
File: ERC20MultiDelegate.sol
45:     constructor(
46:         ERC20Votes _token,
47:         string memory _metadata_uri
48:     ) ERC1155(_metadata_uri) {
49:         token = _token;
50:         //@audit NC - missing event emission for both token and uri
state settings
51:     }
```

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L57C1-L63C6

Missing event emission when internal function finishes execution.

```
File: ERC20MultiDelegate.sol
60:     function delegateMulti(
61:         uint256[] calldata sources,
62:         uint256[] calldata targets,
63:         uint256[] calldata amounts
64:     ) external {
65:         _delegateMulti(sources, targets, amounts);
66:     }
```

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L151C1-L153C6

```
File: ERC20MultiDelegate.sol
160:     function setUri(string memory uri) external onlyOwner {
161:         _setURI(uri);
162:         //@audit NC - missing event emission
163:     }
```

[N-02] Unnecessary typecast to uint256

There is 1 instance of this issue:

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L210

```
File: ERC20MultiDelegate.sol
220:                 uint256(0), // salt //@audit NC - unnecessary
typecast
```

[N-03] Remove unnecessary code

There are 4 instances of this issue:

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L57C1-L63C6

Although following the pattern of external functions calling internal functions is recommended, the delegateMulti() function just calls internal _delegateMulti() function with the passed parameters. This is an unnecessary step. Additionally, there are no complex interactions when calling from the external to internal function, thus making it safe to remove.

Remove this external delegateMulti() function below and make the internal function _delegateMulti() to external.

```
File: ERC20MultiDelegate.sol
60:     function delegateMulti(
61:         uint256[] calldata sources,
62:         uint256[] calldata targets,
63:         uint256[] calldata amounts
64:     ) external {
65:         _delegateMulti(sources, targets, amounts);
66:     }
```

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L86C32-L86C35

Since unsigned integer variables are 0 by default, there is no need to initialize them to 0.

```
86: uint transferIndex = 0;
```

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L198

Remove the parameter ERC20Votes _token from function `retrieveProxyContractAddress`. The function `retrieveProxyContractAddress` is called by functions `_reimburse`, `transferBetweenDelegators` and `deployProxyDelegatorIfNeeded`. These functions always pass the state variable `token` as parameter to the `retrieveProxyContractAddress`. This creates extra gas that can be avoided by just using the state variable `token` in the `retrieveProxyContractAddress` function directly.

Instead of this:

```
File: ERC20MultiDelegate.sol
232:     function retrieveProxyContractAddress(
233:         ERC20Votes _token,
234:         address _delegate
235:     ) private view returns (address) {
236:         bytes memory bytecode = abi.encodePacked(
237:             type(ERC20ProxyDelegator).creationCode,
238:             abi.encode(_token, _delegate)
239:         );
240:         bytes32 hash = keccak256(
241:             abi.encodePacked(
242:                 bytes1(0xff),
243:                 address(this),
244:                 uint256(0), // salt
245:                 keccak256(bytecode)
246:             )
247:         );
248:         return address(uint160(uint256(hash)));
249:     }
```

Use this: **(Note: Make sure to make changes in functions `_reimburse`, `transferBetweenDelegators` and `deployProxyDelegatorIfNeeded` when passing parameters to `retrieveProxyContractAddress`)**

```
File: ERC20MultiDelegate.sol
232:     function retrieveProxyContractAddress(
233:         address _delegate //@audit Now only one parameter
234:     ) private view returns (address) {
235:         bytes memory bytecode = abi.encodePacked(
236:             type(ERC20ProxyDelegator).creationCode,
237:             abi.encode(token, _delegate) //@audit directly using
```

```
state variable "token"
238:         );
239:         bytes32 hash = keccak256(
240:             abi.encodePacked(
241:                 bytes1(0xff),
242:                 address(this),
243:                 uint256(0), // salt
244:                 keccak256(bytecode)
245:             )
246:         );
247:         return address(uint160(uint256(hash)));
248:     }
```

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L189

The return statement `return proxyAddress` below can be removed and a named return can be used. This can additionally save some gas as well.

```
File: contracts/ERC20MultiDelegate.sol
189: return proxyAddress;
```

[N-04] Constants should be defined rather than using magic numbers

There are 2 instances of this issue:

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L186
https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L210

Below are 2 instances of magic number 0 being used for salt. Use a constant variable `salt` to avoid using the magic number directly. This can further increase the readability and maintainability of the code.

```
File: ERC20MultiDelegate.sol
224:  new ERC20ProxyDelegator{salt: 0}(token, delegate);
248:  uint256(0), // salt
```

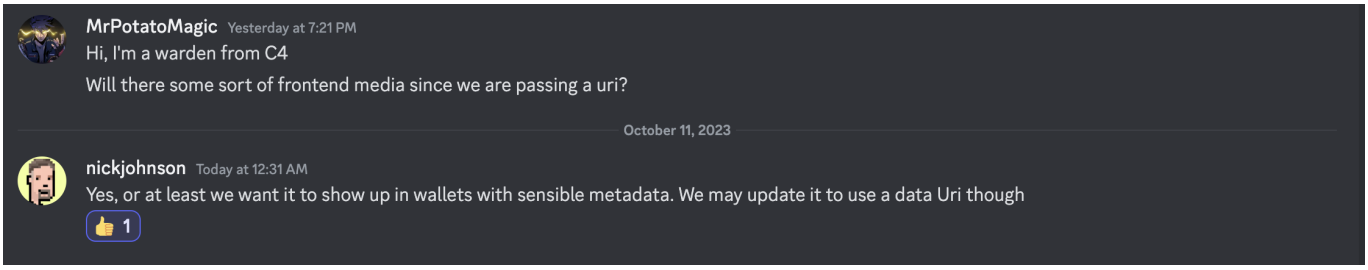
[L-01] Missing getUri() function that returns baseUri appended with a user specific tokenId

Implementing a token type ID substitution mechanism (as mentioned by OpenZeppelin here) is a widespread practice in contracts that use the ERC1155 standard.

This is because instead of the user having to manually append their tokenId themselves, the getUri() function provides it directly when the user provides their tokenId. Additionally, a getUri() function can be helpful to read uri data for a specific tokenId directly from the state since it ensures that the data is upto

date and removes the frontend from the equation to handle the appending of the tokenId. It also serves as an onchain pointer for a specific tokenId that is some metadata stored offchain in the bucket of a IPFS solution.

Small discussion between Sponsor and I which confirms that a data uri will be used to display metadata on the user frontend:



Solution:

```
    function getUri(uint256 tokenId) external view returns(string memory)
{
        string memory tokenID = Strings.toString(tokenId);
        string memory baseUri =  uri(tokenID); //passing tokenID here is
not necessary since it just returns base uri
        return string(abi.encodePacked(baseUri, tokenID));
    }
```

[R-01] Consider adding a time interval field

The time interval field would represent the length of a proposal or the wait time before which delegation cannot be reimbursed or transferred to another delegator.

This can help prevent gaining proposal majority by disallowing a single entity from voting from multiple addresses. This interval is just an extra layer of prevention in case an external voting implementation has an error in it that allows this issue to occur. The interval can be set to 0 for external voting systems that do not require a wait-time.

Gas Optimizations

[G-01] Unnecessary function can be removed

Although following the pattern of external functions calling internal functions is recommended, the delegateMulti() function just calls internal _delegateMulti() function with the passed parameters. This is an unnecessary step. Additionally, there are no complex interactions when calling from the external to internal function, thus making it safe to remove.

There is 1 instance of this:

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L57C1-L63C6

Before VS After

Deployment cost: 4121917 - 4116745 = 5172 gas saved

Function execution cost: 90194 - 90140 = 54 gas saved per call

Remove this external delegateMulti() function below and make the internal function _delegateMulti() to external.

```
File: ERC20MultiDelegate.sol
60:     function delegateMulti(
61:         uint256[] calldata sources,
62:         uint256[] calldata targets,
63:         uint256[] calldata amounts
64:     ) external {
65:         _delegateMulti(sources, targets, amounts);
66:     }
```

[G-02] Use do-while loop instead of for-loop to save gas

There is 1 instance of this:

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L85C1-L108C10

Before VS After

Deployment cost: 4121917 - 4118209 = 3708 gas saved

Function execution cost: 90194 - 89798 = 396 gas saved per call

Instead of this:

```
File: ERC20MultiDelegate.sol
089:         for (
090:             uint transferIndex = 0;//@audit Gas - check if
initialization to 0 can be removed to save gas
091:             transferIndex < Math.max(sourcesLength, targetsLength);
092:             transferIndex++ //@audit Gas - ++i
093:         ) {
094:             address source = transferIndex < sourcesLength
095:                 ? address(uint160(sources[transferIndex]))
096:                 : address(0);
097:             address target = transferIndex < targetsLength
098:                 ? address(uint160(targets[transferIndex]))
099:                 : address(0);
100:             uint256 amount = amounts[transferIndex];
101:
102:             if (transferIndex < Math.min(sourcesLength,
```

```
targetsLength)) {
103:                 // Process the delegation transfer between the
current source and target delegate pair.
104:                 _processDelegation(source, target, amount);
105:             } else if (transferIndex < sourcesLength) {
106:                 // Handle any remaining source amounts after the
transfer process.
107:                 _reimburse(source, amount);
108:             } else if (transferIndex < targetsLength) {
109:                 // Handle any remaining target amounts after the
transfer process.
110:                 createProxyDelegatorAndTransfer(target, amount);
111:             }
112:         }
```

Use this:

```
File: ERC20MultiDelegate.sol
113:         uint256 transferIndex;
114:         do {
115:             address source = transferIndex < sourcesLength
116:                 ? address(uint160(sources[transferIndex]))
117:                 : address(0);
118:             address target = transferIndex < targetsLength
119:                 ? address(uint160(targets[transferIndex]))
120:                 : address(0);
121:             uint256 amount = amounts[transferIndex];
122:
123:             if (transferIndex < Math.min(sourcesLength,
targetsLength)) {
124:                 // Process the delegation transfer between the
current source and target delegate pair.
125:                 _processDelegation(source, target, amount);
126:             } else if (transferIndex < sourcesLength) {
127:                 // Handle any remaining source amounts after the
transfer process.
128:                 _reimburse(source, amount);
129:             } else if (transferIndex < targetsLength) {
130:                 // Handle any remaining target amounts after the
transfer process.
131:                 createProxyDelegatorAndTransfer(target, amount);
132:             }
133:
134:             unchecked {
135:                 ++transferIndex;
136:             }
137:         } while (transferIndex < Math.max(sourcesLength,
targetsLength));
```

[G-03] Remove initialization to 0 to save gas

Since unsigned integer variables are 0 by default, there is no need to initialize them to 0.

There is 1 instance of this issue:

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L86C32-L86C35

Before VS After

Deployment cost: 4121917 - 4121905 = 12 gas saved

```
86: uint transferIndex = 0;
```

[G-04] Remove parameter `ERC20Votes _token` from function `retrieveProxyContractAddress` to save gas

The function `retrieveProxyContractAddress` is called by functions `_reimburse`, `transferBetweenDelegators` and `deployProxyDelegatorIfNeeded`. These functions always pass the state variable `token` as parameter to the `retrieveProxyContractAddress`. This creates extra gas that can be avoided by just using the state variable `token` in the `retrieveProxyContractAddress` function directly.

There is 1 instance of this:

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L198

Before VS After

Deployment cost: 4121917 - 4098816 = 23101 gas saved

Function execution cost: 90194 - 90184 = 10 gas saved per call

Instead of this:

```
File: ERC20MultiDelegate.sol
232:     function retrieveProxyContractAddress(
233:         ERC20Votes _token,
234:         address _delegate
235:     ) private view returns (address) {
236:         bytes memory bytecode = abi.encodePacked(
237:             type(ERC20ProxyDelegator).creationCode,
238:             abi.encode(_token, _delegate)
239:         );
240:         bytes32 hash = keccak256(
241:             abi.encodePacked(
242:                 bytes1(0xff),
243:                 address(this),
244:                 uint256(0), // salt
```

```
245:                 keccak256(bytecode)
246:             )
247:         );
248:         return address(uint160(uint256(hash)));
249:     }
```

Use this: **(Note: Make sure to make changes in functions `_reimburse`, `transferBetweenDelegators` and `deployProxyDelegatorIfNeeded` when passing parameters to `retrieveProxyContractAddress`)**

```
File: ERC20MultiDelegate.sol
232:     function retrieveProxyContractAddress(
233:         address _delegate //@audit Now only one parameter
234:     ) private view returns (address) {
235:         bytes memory bytecode = abi.encodePacked(
236:             type(ERC20ProxyDelegator).creationCode,
237:             abi.encode(token, _delegate) //@audit directly using
state variable "token"
238:         );
239:         bytes32 hash = keccak256(
240:             abi.encodePacked(
241:                 bytes1(0xff),
242:                 address(this),
243:                 uint256(0), // salt
244:                 keccak256(bytecode)
245:             )
246:         );
247:         return address(uint160(uint256(hash)));
248:     }
```

[G-05] Use if conditional statements instead of ternary operators to save gas

There is 1 instance of this:

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L90C1-L96C53

Before VS After

Deployment cost: 4121917 - 4118701 = 3216 gas saved

Function execution cost: 90194 - 90166 = 28 gas saved per call

Instead of this:

```
File: ERC20MultiDelegate.sol
94:             address source = transferIndex < sourcesLength
95:                 ? address(uint160(sources[transferIndex]))
```

```
96:                 : address(0);
97:             address target = transferIndex < targetsLength
98:                 ? address(uint160(targets[transferIndex]))
99:                 : address(0);
```

Use this:

```
File: ERC20MultiDelegate.sol
100:             address source;
101:             address target;
102:             if (transferIndex < sourcesLength) source =
address(uint160(sources[transferIndex]));
103:             if (transferIndex < targetsLength) target =
address(uint160(targets[transferIndex]));
```

[G-06] Use gas-efficient assembly for common math operations like min and max

There are 3 instances of this:

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L80
https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L87
https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L98C39-L98C39

These are the following min and max instances in the code.

```
File: contracts/ERC20MultiDelegate.sol
80: Math.max(sourcesLength, targetsLength) == amountsLength,
87: transferIndex < Math.max(sourcesLength, targetsLength);
98: if (transferIndex < Math.min(sourcesLength, targetsLength)) {
```

The current Math library being used by OpenZeppelin evaluates "max" using ternary operators as follows:

```
    function max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a > b ? a : b;
    }
```

An optimized version for this in assembly would be as follows (Similar to Solady's implementation):

```
    assembly {
        c := xor(a, mul(xor(a, b), gt(b, a)))
    }
```

The reason the above example is more gas efficient is because the ternary operator in the original code contains conditional jumps in the opcodes, which are more costly.

[G-07] Consider using alternatives to OpenZeppelin

Consider using Solmate and Solady. Solmate is a library that provides a number of gas-efficient implementations of common smart contract patterns. Solady is another gas-efficient library that places a strong emphasis on using assembly. In the ERC20MultiDelegate.sol contract, ERC1155 is used frequently for batch minting and burning. Using a gas-optimized version can help reduce gas costs on the user end.

There is 1 instance of this:

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L4C1-L9C54

```
File: ERC20MultiDelegate.sol
4: import {Address} from "@openzeppelin/contracts/utils/Address.sol";
5:
6: import "@openzeppelin/contracts/access/Ownable.sol";
7: import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";
8: import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol";
9: import "@openzeppelin/contracts/utils/math/Math.sol";
```

[G-08] Remove return statement to save gas

There is 1 instance of this:

https://github.com/code-423n4/2023-10-ens/blob/ed25379c06e42c8218eb1e80e141412496950685/contracts/ERC20MultiDelegate.sol#L189

Before VS After

Deployment cost: 4121917 - 4120801 = 1116 gas saved

Function execution cost: 90194 - 90168 = 26 gas saved per call

Instead of this:

```
File: ERC20MultiDelegate.sol
211:     function deployProxyDelegatorIfNeeded(
212:         address delegate
213:     ) internal returns (address) {
214:          address proxyAddress = retrieveProxyContractAddress(token,
```

```
delegate);
215:
216:         // check if the proxy contract has already been deployed
217:         uint bytecodeSize;
218:         assembly {
219:             bytecodeSize := extcodesize(proxyAddress)
220:         }
221:
222:         // if the proxy contract has not been deployed, deploy it
223:         if (bytecodeSize == 0) {
224:             new ERC20ProxyDelegator{salt: 0}(token, delegate);
225:             emit ProxyDeployed(delegate, proxyAddress);
226:         }
227:         return proxyAddress;
228:     }
```

Use this:

```
File: ERC20MultiDelegate.sol
211:     function deployProxyDelegatorIfNeeded(
212:         address delegate
213:     ) internal returns (address proxyAddress) {
214:         proxyAddress = retrieveProxyContractAddress(token, delegate);
215:
216:         // check if the proxy contract has already been deployed
217:         uint bytecodeSize;
218:         assembly {
219:             bytecodeSize := extcodesize(proxyAddress)
220:         }
221:
222:         // if the proxy contract has not been deployed, deploy it
223:         if (bytecodeSize == 0) {
224:             new ERC20ProxyDelegator{salt: 0}(token, delegate);
225:             emit ProxyDeployed(delegate, proxyAddress);
226:         }
227:     }
```

Analysis Report

Comments for the judge to contextualize my findings

This section contains my submitted issues which include Gas/QA reports and certain issues I was researching but turned out to be invalid. Here are the points I would like to point out:

1. Gas Report - My Gas report includes 8 gas optimizations that save 36839 gas in total. None of these gas findings affect the logic of the code, which is proved by all the tests passing. Each finding shows the deployment and function execution cost separately for the sponsor to select the ones that save the most gas without affecting code readability and maintainability.

2. QA Report - My QA report includs 4 non-critical issues, 1 low-severity issue and 1 recommendation. Out of these, the two most important issues are [N-03] and [L-01]. The [N-03] issue points out code that is unnecessary in the codebase and can be optimized to not only improve code readability but also save some gas. The [L-01] issue points out a missing implementation of token type ID substitution mechanism, which is used by user's to read the onchain pointer (baseUri + tokenId) to their metadata stored offchain on an IPFS solution. The other issues submitted as important as well but not as much as these two mentioned above.

Unsubmitted issues:

1. Self-delegation Self-delegation is a common issue found in most voting mechanisms. In this codebase, self-delegation is not prevented. Initially, I thought this could classify as an HM issue but there was no risk associated with this and is considered fine (as mentioned by sponsor). Here is a POC that shows self-delegation is possible (providing in case sponsor wants to add it to the test suite)

- How to add this POC:
    - Run `forge --version` to see if you have foundry installed
    - Run `yarn add --dev @nomicfoundation/hardhat-foundry`
    - Add this `require("@nomicfoundation/hardhat-foundry");` to your hardhat config
    - Run `npx hardhat init-foundry`
    - In the test folder, add `testDelegateMulti.t.sol` and paste the below POC
    - Run `forge test --match-test testUserCanSelfDelegate -vvvvv` to run the test which proves self-delegation is possible

```
File: testDelegateMulti.t.sol
01: // SPDX-License-Identifier: MIT
02: pragma solidity 0.8.19;
03:
04: import {Test, console} from "forge-std/Test.sol";
05: import {ERC20MultiDelegate} from
"../contracts/ERC20MultiDelegate.sol";
06: import {ENSToken} from "../contracts/ENSToken.sol";
07: import {IERC1155Receiver} from
"@openzeppelin/contracts/token/ERC1155/IERC1155Receiver.sol";
08:
09: contract testDelegateMulti is Test {
10:
11:     ENSToken ens;
12:     ERC20MultiDelegate public emd;
13:     address constant USER1 = address(0x01);
14:
15:     uint256[] public  _sources;
16:     uint256[] public _targets;
17:     uint256[] public _amounts;
18:
19:     function setUp() external {
20:         ens = new ENSToken(1e18, 0, 0);
21:         emd = new ERC20MultiDelegate(ens, "");
22:     }
```

```
23:
24:     function testUserCanSelfDelegate() public {
25:         vm.warp(366 days);
26:         //ens.mint(address(this), 10000);
27:         ens.approve(address(emd), 10000);
28:
29:         uint256 currentAddress = uint256(uint160(address(this)));
30:         _targets.push(currentAddress); //UINT256 version of USER1
31:         _targets.push(currentAddress);
32:         _amounts.push(5000);
33:         _amounts.push(5000);
34:
35:         emd.delegateMulti(_sources, _targets, _amounts);
36:
37:         console.log(emd.balanceOf(address(this), currentAddress));
38:         console.log(ens.balanceOf(address(this)));
39:     }
40:
41:     function onERC1155Received(
42:         address operator,
43:         address from,
44:         uint256 id,
45:         uint256 value,
46:         bytes calldata data
47:     ) external returns (bytes4) {
48:         // Handle the received ERC1155 token, and return the expected
function selector
(bytes4(keccak256("onERC1155Received(address,address,uint256,uint256,bytes
)"))).
49:
50:         // Add your custom logic here.
51:
52:         return this.onERC1155Received.selector;
53:     }
54:
55:     function onERC1155BatchReceived(
56:         address operator,
57:         address from,
58:         uint256[] calldata ids,
59:         uint256[] calldata values,
60:         bytes calldata data
61:     ) external returns (bytes4) {
62:         // Handle the received batch of ERC1155 tokens, and return the
expected function selector
(bytes4(keccak256("onERC1155BatchReceived(address,address,uint256[],uint25
6[],bytes)"))).
63:
64:         // Add your custom logic here.
65:
66:         return this.onERC1155BatchReceived.selector;
67:     }
68: }
```

Approach taken in evaluating the codebase

Time spent on this audit: 3 days

Day 1

- Understand how the codebase works
- Mind mapping the codebase architecture and function flow
- QnA with sponsor for doubts
- Add some initial inline bookmarks for surface-level issues not caught by the bot

Day 2

- Researching possible issue leads in codebase
- Add inline bookmarks for Gas and QA issues
- Validate issues bookmarked
- Create reports

Day 3

- Ask sponsor questions on issues that turned out to be invalid
- Write Analysis report

Architecture Recommendations

What's unique?

- ERC1155 for tracking delegations - The ERC1155 standard is usually seen from the perspective of a token. But the sponsor has looked at it from a different perspective of using it to track delegations. This is what makes the code so simplistic in nature and easy to follow.
- Using ProxyDelegator as middleman - The ProxyDelegator contract acts as a middleman instead of locking the tokens in the ERC20MultiDelegate.sol contract itself. This allows each target to be assigned their own unique ProxyDelegator and allow separation of concerns when multiple people delegate using the ERC20MultiDelegate contract. Another reason why this implementation is unique is because it allows proxy-to-proxy transfers, which allows the delegator to transfer voting power from A to B without losing delegation at any point in time. Additionally, the same point of not losing delegation applies when the delegator wants to transfer ERC1155 tokens to another address owned by him.

What's using existing patterns and how this codebase compare to others I'm familiar with

1. ENS VS Canto - Unlike ENS, Canto uses a voting escrow mechanism (inspired by Curve's veCRV model). The approach ENS has taken is quite distinct from Canto. The first difference is that ENS uses ERC1155 for tracking delegations to delegates while Canto uses an in-built `locked` mapping that tracks user struct locks which includes the amount deposited in the lock and the delegate voting power is assigned to (defaulted to user - see here). The second difference is that ENS makes use of ProxyDelegators that store/lock tha ERC20Votes token while Canto uses the VotingEscrow itself to store/lock the CANTO tokens sent as msg.value (see here). The third difference is that ENS allows increasing delegate voting power through the createProxyDelegatorAndTransfer() function while Canto uses a increaseAmount() function which is more complex to understand and requires the user

to increase their lock times. Overall, ENS ERC20MultiDelegate.sol is a much more simplistic and flexible contract that takes on a unique approach to delegations and gets rid of the additional complexity and some features Canto provides.

What ideas can be incorporated?

- Adding a time interval field - This time interval field represents the length of a proposal or the wait time before a user can delegate power to someone else. This can help prevent gaining proposal majority by disallowing a single entity from voting from multiple addresses. This interval is just an extra layer of prevention in case an external voting implementation has an error in it that allows this issue to occur. The interval can be set to 0 for external voting systems that do not require a wait-time.
- Implementing a reward system for honest target delegates - The more the number of users delegate to a target delegate and the longer a user delegates to a target delegate, the greater the rewards for that target delegate for honest behaviour towards its sources.
- Using ERC1155 tokens as LP tokens in a lending/borrowing system, which can serve as proof for backed ERC20Votes tokens in a ProxyDelegator.

Codebase quality analysis

The codebase is high quality due to the difficulty to break it and it's simplicity. The specific aspects of why this codebase is rock solid has been included in sections of this Analysis report such as What's unique? and How this codebase compares to others I'm familiar with.

Additionally, the contract has close to 100% test coverage, which adds an additional layer of security to ensure all functions work as intended. I would recommend the sponsor to integrate a foundry test suite with fuzz tests as well to ensure the functions work with all types of soure-target inputs.

Centralization risks

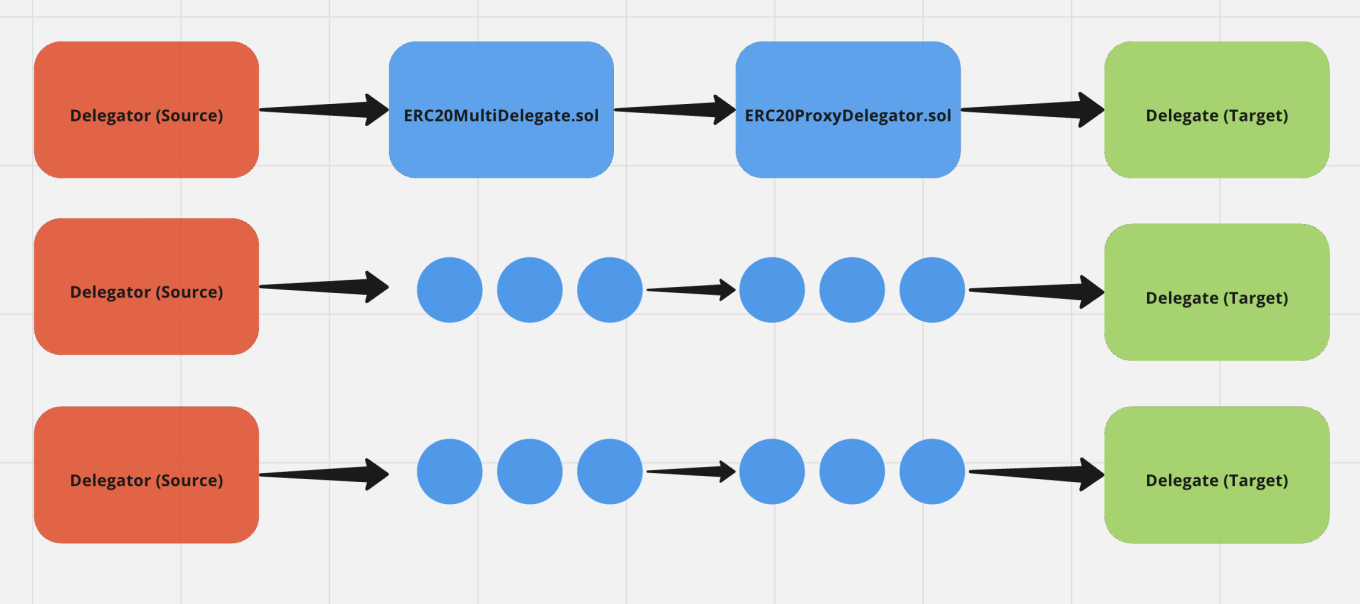
There are no administration roles in this contract, which make it safe to use. The only centralization risk this contract could pose (external to the system) is gaining proposal majority by using the same tokens to vote using different addresses owned by the same entity. This should be mitigated by setting a wait time till the proposal ends before which delegation cannot be reimbursed or transferred to another delegator. This interval/wait-time is just an extra layer of prevention in case an external voting implementation has an error in it that allows this issue to occur. The interval can be set to 0 for external voting systems that do not require a wait-time.

Resources used to gain deeper context on the codebase

1. Contest README - https://github.com/code-423n4/2023-10-ens/blob/main/README.md
2. Understanding ProxyDelegator's creation code - https://blog.openzeppelin.com/deconstructing-a-solidity-contract-part-ii-creation-vs-runtime-6b9d60ecb44c
3. Another resource for contract creation code - https://www.rareskills.io/post/ethereum-contract-creation-code

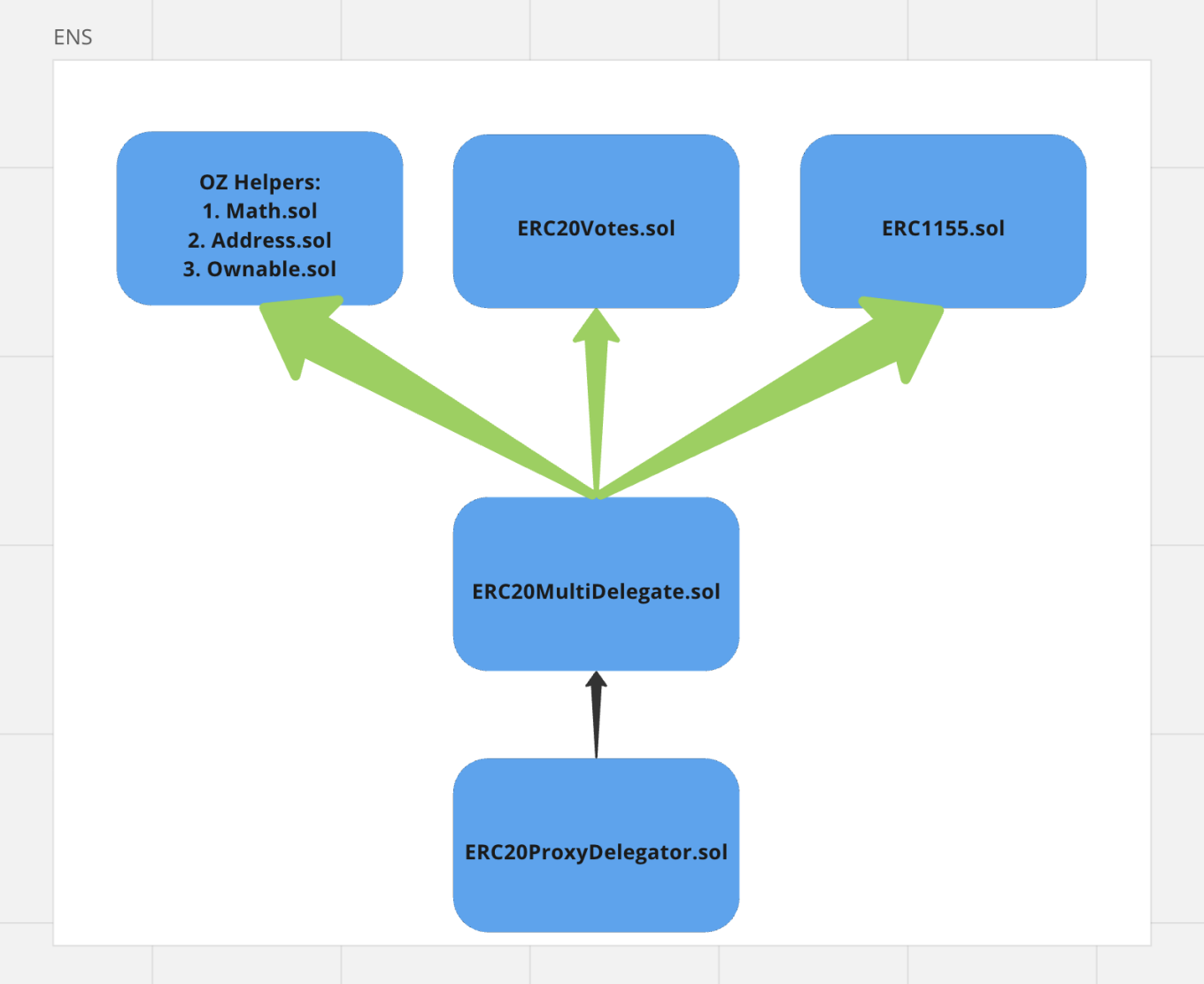
Mechanism Review

High-level System Overview



Documentation/Mental models

Here is the basic inheritance structure of the contracts in scope



Execution Path for different situations in ERC20MultiDelegate.sol contract

A] ERC20MultiDelegate.sol
1] Transfer from user to target delegate directly:
delegateMulti → _delegateMulti → createProxyDelegatorAndTransfer
→ deployProxyDelegatorIfNeeded → transferFrom (token)
→_mintBatch
2] Transfer from target delegate back to user:
delegateMulti → _delegateMulti → _reimburse
→ retrieveProxyContractAddress → transferFrom (token)
→ _burnBatch
3] Transfer from existing delegate to another delegate:
delegateMulti → _delegateMulti → _processDelegation
→ getBalanceForDelegate → deployProxyDelegatorIfNeeded
→ transferBetweenDelegators → retrieveProxyContractAddress
→ transferFrom (token) → _burnBatch → _mintBatch
4] Deployment of a Proxy Delegator:
deployProxyDelegatorIfNeeded → retrieveProxyContractAddress
→ extcodesize check to see if ProxyDelegator exists →
If yes, deploy a new instance (gives ERC20MultiDelegate max approval and delegates token to target delegate) → If not, return existing address

Understanding how array inputs are passed for different situations

The below three instances are the fundamental situations, which can be combined together depending on how inputs are provided.

1. How to get an initial balance to pass this check - Sources are passed as 0 initially, so that createProxyDelegatorAndTransfer is called. This gives the user an ERC1155 balance when processing further delegations and passing the check.
2. How to get reimbursed - For reimbursement, sources are set to the existing target delegates who were previously delegated. This ensures that we enter this if block and execute _burnBatch for those target delegated.
3. How to transfer from existing delegate to another delegate - For delegate-to-delegate delegation, _processDelegation is executed. This transfers the tokens from the respective proxy delegators following which minting and burning occurs for the source and targets with respect to msg.sender (delegator).

User-experience weak spots and how they can be mitigated

1. Optimizing gas costs for users - The current implementation of the contract is solid but it does not consider user experience as one it's core aspects. Since this contract is going to be deployed on Ethereum, optimizing gas costs is a must to allow users to transact more often. This does not pose much of a threat as well since the contract logic is quite simple. My gas report includes most gas optimizations that the sponsor should consider since it does not affect the logic of the code (proved by existing working tests).

Areas of Concern, Attack surfaces, Invariants - QnA

This section includes answers to certain questions/attack surfaces the sponsor wanted addressed.

1. Check for proper permissions and roles
    - ERC1155 tracks source-target delegations in it's mapping, thus every pair is unique and no clashing occurs.
    - ProxyDelegators are unique for each target, thus it is not possible for a target delegate to have overlapping ProxyDelegators
2. Ensure that the delegateMulti function handles array inputs correctly
    - Array inputs are handled correctly in all instances i.e. during typecasting uint256 to address and providing input to ERC20MultiDelegate functions and ERC1155's _mintBatch/_burnBatch functions.
3. Validate the logic for transferring between proxy delegators
    - ERC20Votes tokens are transferred correctly
    - ERC1155 tracking is updated correctly through _mintBatch and _burnBatch

Some questions I asked myself:

1. Are zero address source or delegates possible since the ERC20MultiDelegate contract does not enforce these checks
    - Zero address in sources and delegates are not possible when transferring, minting and burning since 0Z adds address(0) checks and causes reverts if there are any in the arrays.
2. Are retrievals of Proxy contracts and deployment addresses matching?
    - Yes

Time spent:

15 hours