



Frax Ether Liquid Staking contest Findings & Analysis Report

2022-11-29

Table of contents

- [Overview](#)
 - [About C4](#)
 - [Wardens](#)
- [Summary](#)
- [Scope](#)
- [Severity Criteria](#)
- [High Risk Findings \(2\)](#)
 - [\[H-01\] Wrong accounting logic when syncRewards\(\) is called within beforeWithdraw makes withdrawals impossible](#)
 - [\[H-02\] Frontrunning by malicious validator](#)
- [Medium Risk Findings \(10\)](#)
 - [\[M-01\] Centralization risk: admin have privileges: admin can set address to mint any amount of frxETH, can set any address as validator, and change important state in frxETHMinter and withdraw fund from frxETHMinter](#)
 - [\[M-02\] Rewards delay release could cause yields steal and loss](#)
 - [\[M-03\] frxETH can be depegged due to ETH staking balance slashing](#)

- [\[M-04\] removeValidator\(\) and removeMinter\(\) may fail due to exceeding gas limit](#)
- [\[M-05\] frxETHMinter.depositEther may run out of gas, leading to lost ETH](#)
- [\[M-06\] frxETHMinter: Non-conforming ERC20 tokens not recoverable](#)
- [\[M-07\] getNextValidator\(\) error could temporarily make depositEther\(\) inoperable](#)
- [\[M-08\] Withheld ETH should not be sent back to the frxETHMinter contract itself](#)
- [\[M-09\] recoverEther not updating currentWithheldETH breaks calculation of withheld amount for further deposits](#)
- [\[M-10\] sfrxETH: The volatile result of previewMint\(\) may prevent mintWithSignature from working](#)
- [Low Risk and Non-Critical Issues](#)
 - [Low Risk Issues](#)
 - [L-01 Draft OpenZeppelin Dependencies](#)
 - [L-02 Don't use owner and timelock](#)
 - [Non-Critical Issues](#)
 - [N-01 Unused imports](#)
 - [N-02 Non-library/interface files should use fixed compiler versions, not floating ones](#)
 - [N-03 Lint](#)
 - [N-04 Event is missing indexed fields](#)
 - [N-05 Functions, parameters and variables in snake case](#)
 - [N-06 Wrong event parameter name](#)
 - [N-07 Simplify depositWithSignature function](#)
 - [N-08 Missing zero address checks](#)
- [Gas Optimizations](#)
 - [Gas Optimizations Summary](#)

- [G-01 Deleting an array element can use a more efficient algorithm \(1 instance\)](#)
- [G-02 Use function instead of modifiers \(4 instances\)](#)
- [G-03 Use custom errors rather than revert\(\)/require\(\) strings to save gas \(21 instances\)](#)
- [G-04 Using bools for storage incurs overhead \(3 instances\)](#)
- [G-05 Unchecking arithmetics operations that can't underflow/overflow \(7 instances\)](#)
- [G-06 `storage` pointer to a structure is cheaper than copying each value of the structure into `memory`, same for `array` and `mapping` \(1 instance\)](#)
- [G-07 `x = x + y` is more efficient, than `x += y` \(4 instances\)](#)
- [G-08 It costs more gas to initialize non-constant/non-immutable variables to zero than to let the default of zero be applied \(2 instances\)](#)
- [G-09 Don't compare boolean expressions to boolean literals \(3 instances\)](#)
- [G-10 State variables should be cached in stack variables rather than re-reading them from storage \(1 instances\)](#)
- [Overall gas savings](#)

- [Disclosures](#)



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 Frax Ether Liquid Staking smart contract system written in Solidity. The audit contest took place between September 22—September 25 2022.



Wardens

139 Wardens contributed reports to the Frax Ether Liquid Staking contest:

1. [parashar](#)
2. __141345__
3. Lambda
4. [bin2chen](#)
5. Critical
6. [Chom](#)
7. ladboy233
8. [joestakey](#)
9. ronnyx2017
10. rotcivegaf
11. ayeslick
12. rvierdiiev
13. [Trust](#)
14. cccz
15. wagmi
16. [Respx](#)
17. Bahurum
18. Ox1f8b
19. lukris02
20. V_B (Barichek and vlad_bochok)
21. datapunk
22. [Ch_301](#)
23. [oyc_109](#)
24. lllllll
25. Ox4non
26. pashov

27. [pfapostol](#)
28. [bytera](#)
29. [TomJ](#)
30. rbserver
31. [OxNazgul](#)
32. [OxSmartContract](#)
33. BnkeOxO
34. Rolezn
35. neko_nyaa
36. [gogo](#)
37. leosathya
38. ajtra
39. Soosh
40. KIntern_NA (TrungOre and duc)
41. brgLtd
42. [Aymen0909](#)
43. PaludoXO
44. [8olidity](#)
45. peritoflores
46. CodingNameKiki
47. [Sm4rty](#)
48. [JC](#)
49. [rokinot](#)
50. [seyeni](#)
51. [c3phas](#)
52. ReyAdmirado
53. [csanuragjain](#)
54. OptimismSec ([sseefried](#) and [tofunmi](#))
55. bobirichman

- 56. [Deivitto](#)
- 57. cryptostellar5
- 58. Diana
- 59. B2
- 60. [ret2basic](#)
- 61. delfin454000
- 62. RockingMiles (robee and pants)
- 63. Waze
- 64. tnevler
- 65. aysha
- 66. cryptphi
- 67. mics
- 68. [durianSausage](#)
- 69. Triangle (caventa and DeviantArt)
- 70. [Funen](#)
- 71. karanc tf
- 72. [natzuu](#)
- 73. 0x040
- 74. got_targ
- 75. slowmoses
- 76. sach1r0
- 77. asutorufos
- 78. millersplanet
- 79. jag
- 80. Tagir2003
- 81. 0x52
- 82. yixxas
- 83. 0xf15ers (remora and twojoy)
- 84. [a12jmx](#)

- 85. sikorico
- 86. JLevick
- 87. bbuddha
- 88. [yasir](#)
- 89. yongskiws
- 90. [obront](#)
- 91. Yiko
- 92. Tointer
- 93. [exd0tpy](#)
- 94. [bharg4v](#)
- 95. [prasantgupta52](#)
- 96. Ox5rings
- 97. SnowMan
- 98. ch0bu
- 99. peanuts
- 100. [medikko](#)
- 101. [zishansami](#)
- 102. [Rohan16](#)
- 103. erictee
- 104. d3e4
- 105. RaymondFam
- 106. OxA5DF
- 107. [Tomio](#)
- 108. Amithuddar
- 109. Metatron
- 110. samruna
- 111. drdr
- 112. bulej93
- 113. [Satyam_Sharma](#)

114. [Ocean_Sky](#)

115. imare

116. JAGADESH

117. SooYa

118. Pheonix

119. [Fitraldys](#)

120. Oxsam

121. [fatherOfBlocks](#)

122. [albincsergo](#)

123. beardofginger

124. Ben

125. emrekocak

126. [dharma09](#)

127. Oxmatt

128. OxSky

129. [hansfrieze](#)

130. m9800

131. [magu](#)

132. [pedroais](#)

133. [Ruhum](#)

This contest was judged by [Oxean](#).

Final report assembled by [itsmetechjay](#).



Summary

The C4 analysis yielded an aggregated total of 12 unique vulnerabilities. Of these vulnerabilities, 2 received a risk rating in the category of HIGH severity and 10 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 83 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 93 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 Frax Ether Liquid Staking contest repository](#), and is composed of 6 smart contracts written in the Solidity programming language and includes 413 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 (2)



[H-01] Wrong accounting logic when syncRewards() is called within beforeWithdraw makes withdrawals impossible

Submitted by Lambda, also found by bin2chen and Critical

`sfrxETH.beforeWithdraw` first calls the `beforeWithdraw` of `xERC4626`, which decrements `storedTotalAssets` by the given amount. If the timestamp is greater than the `rewardsCycleEnd`, `syncRewards` is called. However, the problem is that the assets have not been transferred out yet, meaning `asset.balanceOf(address(this))` still has the old value. On the other hand, `storedTotalAssets` was already updated. Therefore, the following calculation will be inflated by the amount for which the withdrawal was requested:

```
uint256 nextRewards = asset.balanceOf(address(this)) - storedTot
```

This has severe consequences:

1. During the following reward period, `lastRewardAmount` is too high, which means that too many rewards are paid out to users who want to withdraw. A user could exploit this to steal the assets of other users.
2. When `syncRewards()` is called the next time, it is possible that the `nextRewards` calculation underflows because `lastRewardAmount > asset.balanceOf(address(this))`. This is very bad because `syncRewards()` will be called in every withdrawal (after the `rewardsCycleEnd`) and none of them will succeed because of the underflow. Depositing more also does not help here, it just increases `asset.balanceOf(address(this))` and `storedTotalAssets` by the same amount, which does not eliminate the underflow.

Note that this bug does not require a malicious user or a targeted attack to surface. It can (and probably will) happen in practice just by normal user interactions with the vault (which is for instance shown in the PoC).



Proof Of Concept

Consider the following test:

```
function testTotalAssetsAfterWithdraw() public {
    uint128 deposit = 1 ether;
    uint128 withdraw = 1 ether;
    // Mint frxETH to this testing contract from nothing, fo
    mintTo(address(this), deposit);
```

```

        // Generate some sfrxETH to this testing contract using
        sfrxETHtoken.approve(address(sfrxETHtoken), deposit);
        sfrxETHtoken.deposit(deposit, address(this));
        require(sfrxETHtoken.totalAssets() == deposit);

        vm.warp(block.timestamp + 1000);
        // Withdraw frxETH (from sfrxETH) to this testing contract
        sfrxETHtoken.withdraw(withdraw, address(this), address(this));
        vm.warp(block.timestamp + 1000);
        sfrxETHtoken.syncRewards();
        require(sfrxETHtoken.totalAssets() == deposit - withdraw);
    }
}

```

This is a normal user interaction where a user deposits into the vault, and makes a withdrawal some time later. However, at this point the `syncRewards()` within the `beforeWithdraw` is executed. Because of that, the documented accounting mistake happens and the next call (in fact every call that will be done in the future) to `syncRewards()` reverts with an underflow.



Recommended Mitigation Steps

Call `syncRewards()` before decrementing `storedTotalAssets`, i.e.:

```

function beforeWithdraw(uint256 assets, uint256 shares) internal {
    if (block.timestamp >= rewardsCycleEnd) { syncRewards(); }
    super.beforeWithdraw(assets, shares); // call xERC4626's
}

```

Then, `asset.balanceOf(address(this))` and `storedTotalAssets` are still in sync within `syncRewards()`.

FortisFortuna (Frax) commented:

Does this only occur if all users try to withdraw at the exact same time? If so, this is a known bug by us and the risk would be low in a real-life deployment scenario. We can also let the users know about the ramping of the rewards.

FortisFortuna (Frax) marked as duplicate

[Lambda \(warden\) commented:](#)

I do not think that this is a duplicate of [#311](#). #311 (and the other issues that are linked there) describe a recoverable issue where the withdrawal for the last user fails (which was listed as a known issue of xERC4626) until the cycle ends.

The issue here that is described here and demonstrated in the PoC is a non-recoverable sfrxETH-specific issue (because sfrxETH potentially calls `syncRewards()` in the `beforeWithdraw` function) where withdrawals even fail after the cycle has ended. It also does not require all users to withdraw at the same time.

[FortisFortuna \(Frax\) commented:](#)

@Lambda What about [24](#) ?

[Lambda \(warden\) commented:](#)

@FortisFortuna Good catch did not see that, yes [24](#) addresses the same issue

[FortisFortuna \(Frax\) confirmed and commented:](#)

@Lambda I will mark yours as primary because it is better documented.

[corddry \(Frax\) commented:](#)

Here's the proposed fix, which instead moves the `syncRewards` call to a modifier, so that it actually occurs `_before` the `_withdraw__` instead of in `beforeWithdraw`. It also adds it to the other 4626 `withdraw/redeem` functions. Would appreciate feedback if you have any

<https://github.com/FraxFinance/frxETH-public/pull/2/commits/1ec457c7f5faed618971fb29b9bcc6d54453b093>

[Lambda \(warden\) commented:](#)

The modifier is currently missing for `mint` and `redeem`, is that on purpose? Otherwise, it looks good to me

[corddry \(Frax\) commented:](#)

Whoops— nice catch, added here <https://github.com/FraxFinance/frxETH-public/commit/996d528b46d1b2a0ac2e5b8f6d2138ccab8e03f5>



[H-02] Frontrunning by malicious validator

Submitted by parashar

Frontrunning by malicious validator changing withdrawal credentials.



Proof of Concept

A malicious validator can frontrun depositEther transaction for its pubKey and deposit 1 ether for different withdrawal credential, thereby setting withdrawal credit before deposit of 32 ether by contract and thereby when 32 deposit ether are deposited, the withdrawal credential is also what was set before rather than the one being sent in depositEther transaction.



Recommended Mitigation Steps

Set withdrawal credentials for validator by depositing 1 ether with desired withdrawal credentials, before adding it in Operator Registry.

[FortisFortuna \(Frax\) commented:](#)

Interesting point, but at the beginning, the only validators we will have will be Frax controlled.

[Oxean \(judge\) commented:](#)

```
function deposit(
    bytes calldata pubkey,
    bytes calldata withdrawal_credentials,
    bytes calldata signature,
    bytes32 deposit_data_root
) override external payable {
    // Extended ABI length checks since dynamic types are used
    require(pubkey.length == 48, "DepositContract: invalid p
```

```

require(withdrawal_credentials.length == 32, "DepositContract: invalid withdrawal_credentials")
require(signature.length == 96, "DepositContract: invalid signature")

// Check deposit amount
require(msg.value >= 1 ether, "DepositContract: deposit amount too small")
require(msg.value % 1 gwei == 0, "DepositContract: deposit amount not a multiple of 1 gwei")
uint deposit_amount = msg.value / 1 gwei;
require(deposit_amount <= type(uint64).max, "DepositContract: deposit amount too large")

// Emit `DepositEvent` log
bytes memory amount = to_little_endian_64(uint64(deposit_amount));
emit DepositEvent(
    pubkey,
    withdrawal_credentials,
    amount,
    signature,
    to_little_endian_64(uint64(deposit_count))
);

// Compute deposit data root (`DepositData` hash tree root)
bytes32 pubkey_root = sha256(abi.encodePacked(pubkey, bytes32(0)));
bytes32 signature_root = sha256(abi.encodePacked(
    sha256(abi.encodePacked(signature[:64])),
    sha256(abi.encodePacked(signature[64:], bytes32(0)))
));
bytes32 node = sha256(abi.encodePacked(
    sha256(abi.encodePacked(pubkey_root, withdrawal_credentials)),
    sha256(abi.encodePacked(amount, bytes24(0), signature_root))
));

// Verify computed and expected deposit data roots match
require(node == deposit_data_root, "DepositContract: root mismatch")

// Avoid overflowing the Merkle tree (and prevent edge case of zero height)
require(deposit_count < MAX_DEPOSIT_COUNT, "DepositContract: deposit count too large")

// Add deposit data root to Merkle tree (update a single leaf)
deposit_count += 1;
uint size = deposit_count;
for (uint height = 0; height < DEPOSIT_CONTRACT_TREE_DEPTH; height++) {
    if ((size & 1) == 1) {
        branch[height] = node;
        return;
    }
    node = sha256(abi.encodePacked(branch[height], node));
    size /= 2;
}

```

```
}  
// As the loop should always end prematurely with the `r  
// this code should be unreachable. We assert `false` ju  
assert(false);  
}
```

Oxean (judge) commented:

It is unclear both in the code above for the deposit contract as well as the documentation on keys

<https://kb.beaconcha.in/ethereum-2.0-depositing>

<https://kb.beaconcha.in/ethereum-2-keys>

How exactly multiple deposits two the same validator using different withdrawal keys would work. While it would make sense that they would allow a one to many mapping, I am unable to confirm or deny this and therefore will leave the risk currently as High on the side of caution.

Trust (warden) commented:

Strong find. Indeed in ETH [specs](#) we can see that in `process_deposit()`, if the pubkey is already registered, we just increase its balance, not touching the `withdrawal_credentials`. However the recommended mitigation does not really address the issue IMO, and the detail is quite lacking.

FortisFortuna (Frax) commented:

I think it is technically a non-issue because we will be controlling the addition/removal of validators. Should that eventually become open, we will have to look at the entire code from a different perspective to close security holes.

Trust (warden) commented:

I think it is relevant, because the idea is to make the protocol controlled validators work for the attacker, because they inserted their own withdrawal credentials directly on the deposit contract.

FortisFortuna (Frax) confirmed and commented:

Ohh I see it now. Good point.

More info

<https://research.lido.fi/t/mitigations-for-deposit-front-running-vulnerability/1239>

Since all of the validators are ours and we have the mnemonic, would it still be an issue though? Lido's setup is different:

<https://medium.com/immunefi/rocketpool-lido-frontrunning-bug-fix-postmortem-e701f26d7971>

[FortisFortuna \(Frax\) commented:](#)

<https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#deposits>

From @OxJM

In the scenario that someone frontruns us with a 1 ETH deposit at the same time we do a 32 ETH deposit, their 1 ETH deposit would fail on beaconchain because it would fail bls.Verify. The result would be them losing their 1 ETH.

Our 32 ETH would go through normally and the validator would activate

[Oxean \(judge\) commented:](#)

@FortisFortuna - can you elaborate on why you believe that bls.Verify would fail?

```
if not bls.Verify(pubkey, signing_root, deposit.data.signature):
```

[FortisFortuna \(Frax\) commented:](#)

From @OxJM

https://github.com/ethereum/staking-deposit-cli/blob/e2a7c942408f7fc446b889097f176238e4a10a76/staking_deposit/credentials.py#L127

the signing root includes the deposit message which has the withdrawal credentials

https://github.com/ethereum/staking-deposit-cli/blob/e2a7c942408f7fc446b889097f176238e4a10a76/staking_deposit_credentials.py#L112

hence `bls.Verify` would fail on Beaconchain as I mentioned

the consensus spec has that $\text{signingroot} = \text{computesigningroot}(\text{depositmessage}, \text{domain})$ which is verified against the signature.

[Oxean \(judge\) commented:](#)

The signature would be valid. The validator would still sign the message containing the credentials that they are front running with.

[FortisFortuna \(Frax\) commented:](#)

From @denett

“The signature would be valid. The validator would still sign the message containing the credentials that they are front running with.” Only the validator can create a valid signature and we own the key to the validator.

[Oxean \(judge\) commented:](#)

Yea, so this is the root of it, the contest does not specify that Frax is the owner of all validators that are meant to be used with this protocol. Without stating that ahead of time for the Wardens to understand, I believe this to be a valid finding and the warden should be awarded.

[FortisFortuna \(Frax\) commented:](#)

Ok. So in our current setup, assuming Frax owns all validators, we are safe?

[Oxean \(judge\) commented:](#)

:) I cannot guarantee anything in DeFi is safe. My understanding of this particular vulnerability is that it would require a validator to act maliciously by using a smaller than 32 ETH deposit to front run your deposit and enable them to control the withdrawal in the future. If the validator is owned by your team and the keys are never exploited, then I don't see how the front ran signature could be generated.

[FortisFortuna \(Frax\) commented:](#)

Ya, I hear you lol. At least for this particular scenario we are ok then, according to the known bug. We can pay out for the bug because none of our team were aware of it and it is good to know for the future.



Medium Risk Findings (10)



[M-01] Centralization risk: admin have privileges: admin can set address to mint any amount of frxETH, can set any address as validator, and change important state in frxETHMinter and withdraw fund from frcETHMinter

Submitted by ladboy233, also found by OxSmartContract, Solidity, ayeslick, Aymen0909, cccz, Chom, csanuragjain, llllll, joestakey, neko_nyaa, OptimismSec, PaludoXO, pashov, peritoflores, rbserver, rvierdiev, and TomJ

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/ERC20/ERC20PermitPermissionedMint.sol#L41>

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/ERC20/ERC20PermitPermissionedMint.sol#L53>

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/ERC20/ERC20PermitPermissionedMint.sol#L65>

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/ERC20/ERC20PermitPermissionedMint.sol#L76>

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/ERC20/ERC20PermitPermissionedMint.sol#L94>

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/frxETHMinter.sol#L159>

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/frxETHMinter.sol#L166>

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/frxETHMinter.sol#L177>

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/frxETHMinter.sol#L184>

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/frxETHMinter.sol#L191>

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/frxETHMinter.sol#L199>

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/OperatorRegistry.sol#L53>

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/OperatorRegistry.sol#L61>

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/OperatorRegistry.sol#L69>

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/OperatorRegistry.sol#L82>

<https://github.com/code-423n4/2022-09->

[frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/OperatorRegistry.sol#L93](#)



Impact

Admin have privileges: admin can set address to mint any amount of frxETH, can set any address as validator, and change important state in frxETHMinter and withdraw fund from frxETHMinter.

Note the modifier below, either the timelock governance contract or the contract owner can access to all the high privilege function.

```
modifier onlyByOwnGov() {
    require(msg.sender == timelock_address || msg.sender ==
    _);
}
```

There are numerous methods that the admin could apply to rug pull the protocol and take all user funds.

The admin can

add or remove validator from OperatorRegistry.sol

set minter address or remove minter address in frxETH.sol

minter set by admin can mint or burn any amount of frxETH token.

set ETE deduction ratio, withdraw any amount of ETH or ERC20 token



Tools Used

Foundry



Recommended Mitigation Steps

Without significant redesign it is not possible to avoid the admin being able to rug pull the protocol.

As a result the recommendation is to set all admin functions behind either a timelocked DAO or at least a timelocked multisig contract.

FortisFortuna (Frax) commented:

We are well aware of the permission structure. The owner will most likely be a large multisig. We mentioned the Frax Multisig in the scope too.

Oxean (judge) commented:

Going to use this issue as the canonical issue for all “malicious owner” type reports. The protocol does have some serious “trust” in the administrator and the highlighted issues are important for end users to understand and should be part of the report.



[M-02] Rewards delay release could cause yields steal and loss

Submitted by __141345__, also found by Bahurum, Ch_301, Chom, datapunk, Respx, ronnyx2017, and Trust

In the current rewards accounting, vault shares in `deposit()` and `redeem()` can not correctly record the spot yields generated by the staked asset. Yields are released over the next rewards cycle. As a result, malicious users can steal yields from innocent users by picking special timing to `deposit()` and `redeem()`.



Proof of Concept

In `syncRewards()`, the current asset balance is broken into 2 parts:

`storedTotalAssets` and `lastRewardAmount/nextRewards`. The

`lastRewardAmount` is the surplus balance of the asset, or the most recent yields.

```
// lib/ERC4626/src/xERC4626.sol
function syncRewards() public virtual {
    // ...

    uint256 nextRewards = asset.balanceOf(address(this)) - s
```

```

        storedTotalAssets = storedTotalAssets_ + lastRewardAmount;

        uint32 end = ((timestamp + rewardsCycleLength) / rewardsCycleLength);

        lastRewardAmount = nextRewards.safeCastTo192();
        // ...
        rewardsCycleEnd = end;
    }

```

And in the next rewards cycle, `lastRewardAmount` will be linearly added to `storedTotalAssets`, their sum is the return value of `totalAssets()`:

```

function totalAssets() public view override returns (uint256) {
    // ...

    if (block.timestamp >= rewardsCycleEnd_) {
        // no rewards or rewards fully unlocked
        // entire reward amount is available
        return storedTotalAssets_ + lastRewardAmount_;
    }

    // rewards not fully unlocked
    // add unlocked rewards to stored total
    uint256 unlockedRewards = (lastRewardAmount_ * (block.timestamp - rewardsCycleEnd_)) / rewardsCycleLength;
    return storedTotalAssets_ + unlockedRewards;
}

```

`totalAssets()` will be referred when `deposit()` and `redeem()`.

```

// lib/solmate/src/mixins/ERC4626.sol

```

```

function deposit(uint256 assets, address receiver) public virtual returns (uint256 shares) {
    require((shares = previewDeposit(assets)) != 0, "ZERO_SHARES");
    // ...
    _mint(receiver, shares);
    // ...
}

```

```

function redeem() public virtual returns (uint256 assets) {
    // ...
    require((assets = previewRedeem(shares)) != 0, "ZERO_ASSETS");
}

```

```

        beforeWithdraw(assets, shares);

        _burn(owner, shares);

        // ...

        asset.safeTransfer(receiver, assets);
    }

    function previewDeposit(uint256 assets) public view virtual
        return convertToShares(assets);
    }

    function previewRedeem(uint256 shares) public view virtual
        return convertToAssets(shares);
    }

    function convertToShares(uint256 assets) public view virtual
        uint256 supply = totalSupply;

        return supply == 0 ? assets : assets.mulDivDown(supply,

    }

    function convertToAssets(uint256 shares) public view virtual
        uint256 supply = totalSupply;

        return supply == 0 ? shares : shares.mulDivDown(totalAss
    }

```

Based on the above rules, there are 2 potential abuse cases:

1. If withdraw just after the `rewardsCycleEnd` timestamp, a user can not get the yields from last rewards cycle. Since the `totalAssets()` only contain `storedTotalAssets` but not the yields part. It takes 1 rewards cycle to linearly add to the `storedTotalAssets`.

Assume per 10,000 asset staking generate yields of 70 for 7 days, and the reward cycle is 1 day. A malicious user Alice can do the following:

- Watch the mempool for `withdraw(10,000)` from account Bob, front run it with `syncRewards()`, so that the most recent yields of amount 70 from Bob will stay

in the vault.

- Alice will also deposit a 10,000 to take as much shares as possible.
- After 1 rewards cycle of 1 day, `redeem()` to take the yields of 70.

Effectively steal the yields from Bob. The profit for Alice is not 70, because after 1 day, her own deposit also generates some yield, in this example this portion is 1. At the end, Alice steal yield of amount 60.

2. When the Multisig Treasury transfers new yields into the vault, the new yields will accumulate until `syncRewards()` is called. It is possible that yields from multiple rewards cycles accumulates, and being released in the next cycle.

Knowing that the yields has been accumulated for 3 rewards cycles, a malicious user can `deposit()` and call `syncRewards()` to trigger the release of the rewards. `redeem()` after 1 cycle.

Here the malicious user gets yields of 3 cycles, lose 1 in the waiting cycle. The net profit is 2 cycle yields, and the gained yields should belong to the other users in the vault.



Recommended Mitigation Steps

- For the `lastRewardAmount` not released, allow the users to redeem as it is linearly released later.
- For the accumulated yields, only allow users to redeem the yields received after 1 rewards cycle after the deposit.

[FortisFortuna \(Frax\) confirmed, but disagreed with severity and commented:](#)

From @denett

`syncRewards` should be called by us at the beginning of each period, or we need to automatically call it before deposits/withdrawals.

[Oxean \(judge\) commented:](#)

All of the duplicated issues reference a scenario where `syncRewards` isn't called at the appropriate time leading to the ability for users to steal yield from other users

in some fashion. So while they are slightly different I do think grouping them together makes sense as the underlying root cause is the same.

Medium seems like the appropriate severity for this, as it requires some external factors and doesn't result in principal being lost, only yield.



[M-03] frxETH can be depegged due to ETH staking balance slashing

Submitted by ladboy233, also found by __141345__

The main risk in ETH 2.0 POS staking is the slashing penalty, in that case the frxETH will not be pegged and the validator cannot maintain a minimum 32 ETH staking balance.

<https://cryptobriefing.com/ethereum-2-0-validators-slashed-staking-pool-error/>



Recommended Mitigation Steps

We recommend the protocol to add mechanism to ensure the frxETH is pegged via burning if case the ETH got slashed.

And consider when the node does not maintain a minimum 32 ETH staking balance, who is in charge of adding the ETH balance to increase the staking balance or withdraw the ETH and distribute the fund.

[FortisFortuna \(Frax\) commented:](#)

We as the team can either choose to subsidize this, or let it float. ETH 2.0 does not allow unstaking yet. When it eventually does, we will redeploy this minting contract with updated logic that may be helpful.

[Oxean \(judge\) decreased severity to Medium and commented:](#)

I think this is valid but should be downgraded to Medium. Users should be aware that there is no mechanism built in to deal with slashing and that the asset backed guarantee isn't without some (perhaps negligible) risk of slashing.



[M-04] removeValidator() and removeMinter() may fail due to exceeding gas limit

Submitted by oyc_109, also found by Ox4non, Chom, ladboy233, Lambda, lukris02, pashov, Respx, and V_B

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/OperatorRegistry.sol#L113-L118>

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/ERC20/ERC20PermitPermissionedMint.sol#L84-L89>



Vulnerability Details



removeValidator() and removeMinter() may fail due to exceeding gas limit

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/OperatorRegistry.sol#L113-L118>

```
for (uint256 i = 0; i < original_validators.length;
    if (i != remove_idx) {
        validators.push(original_validators[i]);
    }
}
```

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/ERC20/ERC20PermitPermissionedMint.sol#L84-L89>

```
for (uint i = 0; i < minters_array.length; i++){
    if (minters_array[i] == minter_address) {
        minters_array[i] = address(0); // This will leave
        break;
    }
}
```

The `removeValidator()` is used to remove a validator from the array `validators`.

There is an unbounded loop in `removeValidator()` such that if the `validators` array gets sufficiently large, this function call will fail due to exceeding the gas limit.

The same issue exists in the `removeMinter()` function. If `minters_array` gets large, the function call will fail.

FortisFortuna (Frax) commented:

Technically correct, but in practice, the number of minters will always remain low. If it becomes an issue, we can designate one minter as a “pre-minter” that has a batch of tokens minted to it beforehand, then auxiliary contracts can connect to that instead of ERC20PermitPermissionedMint.sol instead.

Oxean (judge) commented:

I think Medium is appropriate here, given this could impact the functionality of the protocol.

Trust (warden) commented:

Wouldn't call this a risk to the functionality of the protocol, because sender can always send enough gas, and validator array gets truncated every time on is popped for use.

Unbounded for-loops should be handled with care but not sure a realistic impact can be demonstrated here to qualify for Medium.



[M-05] frxETHMinter.depositEther may run out of gas, leading to lost ETH

Submitted by Lambda, also found by Ox52, Bahurum, BnkeOx0, KIntern_NA, lukris02, rbserver, Respx, rotcivegaf, Soosh, TomJ, Trust, V_B, and yixxas

`frxETHMinter.depositEther` always iterates over all deposits that are possible with the current balance (`(address(this).balance - currentWithheldETH) /`

`DEPOSIT_SIZE`). However, when a lot of ETH was deposited into the contract / it was not called in a long time, this loop can reach the gas limit. When this happens, no more calls to `depositEther` are possible, as it will always run out of gas.

Of course, the probability that such a situation arises depends on the price of ETH. For >1,000 USD it would require someone to deposit a large amount of money (which can also happen, there are whales with thousands of ETH, so if one of them would decide to use frxETH, the problem can arise). For lower prices, it can happen even for small (in dollar terms) deposits. And in general, the correct functionality of a protocol should not depend on the price of ETH.



Proof Of Concept

Jerome Powell continues to raise interest rates, he just announced the next rate hike to 450%. The crypto market crashes, ETH is at 1 USD. Bob buys 100,000 ETH for 100,000 USD and deposits them into `frxETHMinter` . Because of this deposit, `numDeposit` within `depositEther` is equal to 3125. Therefore, every call to the function runs out of gas and it is not possible to deposit this ETH into the deposit contract.



Recommended Mitigation Steps

It should be possible to specify an upper limit for the number of deposits such that progress is possible, even when a lot of ETH was deposited into the contract.

[FortisFortuna \(Frax\) confirmed, but decreased severity to Low and commented:](#)



Adding a `maxLoops` parameter or similar can help mitigate this for sure.

[Oxean \(judge\) increased severity to Medium and commented:](#)



Warden(s) fail to demonstrate how this leads to a loss of funds which would be required for High Severity. This does however lead directly to emergency failover's having to be called to remove the now stuck ETH, and ultimately impairs the functionality and availability of the protocol, so Medium severity is appropriate.



[M-06] frxETHMinter: Non-conforming ERC20 tokens not recoverable

Submitted by Lambda, also found by 0x1f8b, 0x5rings, 0xSky, 0xSmartContract, 8olidity, brgltd, Chom, CodingNameKiki, hansfrieze, llllll, m9800, magu, pashov, pedroais, peritoflores, prasantgupta52, rokinot, Ruhum, seyni, and Sm4rty

There is a function `recoverERC20` to rescue any ERC20 tokens that were accidentally sent to the contract. However, there are tokens that do not return a value on success, which will cause the call to revert, even when the transfer would have been successful. This means that those tokens will be stuck forever and not be recoverable.



Proof Of Concept

Someone accidentally transfers USDT, one of the most commonly used ERC20 tokens, to the contract. Because USDT's transfer [does not return a boolean](#), it will not be possible to recover those tokens and they will be stuck forever.



Recommended Mitigation Steps

Use OpenZeppelin's `safeTransfer`.

[FortisFortuna \(Frax\) commented:](#)

Not really medium risk. Technically you could use `safeTransfer`, but if someone were to accidentally send something to this contract, it would most likely be either ETH, FRAX, frxETH, or sfrxETH, all of which are transfer compliant.

[Oxean \(judge\) commented:](#)

I think this qualifies as a Medium risk. Sponsor has created functionality to recover ERC20 tokens. Wardens have shown a path to which this functionality does not work correctly.

2 — Med: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.



[M-07] getNextValidator() error could temporarily make depositEther() **inoperable**

Submitted by __141345__

When depositEther() , if 1 validators is used before, the whole deposit function will revert, causing DoS. depositEther() function will be inoperable until the gov manually removes the mistaken validator.



Proof of Concept

In depositEther() , if the pubKey is already used, the whole loop will revert, and the deposit operation cannot move on.

```
// src/frxETHMinter.sol
function depositEther() external nonReentrant {
    // ...

    for (uint256 i = 0; i < numDeposits; ++i) {
        // Get validator information
        (
            bytes memory pubKey,
            bytes memory withdrawalCredential,
            bytes memory signature,
            bytes32 depositDataRoot
        ) = getNextValidator(); // Will revert if there are

        // Make sure the validator hasn't been deposited int
        // until withdrawals are allowed
        require(!activeValidators[pubKey], "Validator already used")
    }
    // ...
}
```

And in the next rewards cycle, lastRewardAmount will be linearly added to storedTotalAssets , their sum is the return value of totalAssets() :

```
function totalAssets() public view override returns (uint256) {
    // ...
}
```

```

    if (block.timestamp >= rewardsCycleEnd_) {
        // no rewards or rewards fully unlocked
        // entire reward amount is available
        return storedTotalAssets_ + lastRewardAmount_;
    }

    // rewards not fully unlocked
    // add unlocked rewards to stored total
    uint256 unlockedRewards = (lastRewardAmount_ * (block.timestamp - rewardsCycleEnd_) / rewardsCycleLength_);
    return storedTotalAssets_ + unlockedRewards;
}

```

Temporarily the `depositEther()` function will be inaccessible. Until the governance calls the registry to pop the wrong validator.

```

// src/OperatorRegistry.sol
function popValidators(uint256 times) public onlyByOwnGov {
    // Loop through and remove validator entries at the end
    for (uint256 i = 0; i < times; ++i) {
        validators.pop();
    }

    emit ValidatorsPopped(times);
}

```



Recommended Mitigation Steps

Use `try/catch` to skip the wrong validator, then the deposit function will be more robust to unexpected situations.

FortisFortuna (Frax) commented:

We plan to keep an eye on the number of free validators and have a decent sized buffer of them.

Oxean (judge) commented:

Awarding as Medium, given that this can disable deposits, the registry should check against the mapping.



[M-08] Withheld ETH should not be sent back to the frxETHMinter contract itself

Submitted by ronnyx2017, also found by ayeslick and rvierdiiev

<https://github.com/code-423n4/2022-09-frax/blob/main/src/frxETHMinter.sol#L166-L174>

<https://github.com/code-423n4/2022-09-frax/blob/main/src/frxETHMinter.sol#L191-L196>

<https://github.com/code-423n4/2022-09-frax/blob/main/src/frxETHMinter.sol#L114-L116>



Impact

It will lead to duplicating accounting for the Eths which have been already converted to the frxETH tokens. It means Eth:frxEth will not be 1:1, and eventually leads to decoupling.



Proof of Concept

The function `moveWithheldETH` will send the `amount` of the Withheld ETH in the contract to the address `to`. It doesn't check if the `to` address is the frxETHMinter contract itself.

And the frxETHMinter has the receive function which will submit any eth received to the frxETH.

```
/// @notice Fallback to minting frxETH to the sender
receive() external payable {
    _submit(msg.sender);
}
```

But these parts of Eths (WithheldETH) also have been converted to the frxETH normally when they were sent to the contract at the first time.

```
function _submit(address recipient) internal nonReentrant {
```



```
// Initial pause and value checks
...
// Give the sender frxETH
frxETHToken.minter_mint(recipient, msg.value);
```

So these Eths will be accounted, Twice, even more. It means Eth:frxEth will not be 1:1 anymore.

The function `recoverEther` has the same problem. Although these two functions can only be called by owner or DAO gov. It seriously affects financial stability.



Recommended Mitigation Steps

Furthermore, due to the logic `receive() -> submit()`, any kind of transaction that withdraws ETH from the contract and then sends it back will cause the same problem.

A non-feedback payable empty function that does not use `_submit()` should be added to receive special ETH without increasing the frxeth supply.

FortisFortuna (Frax) commented:

We are well aware of the permission structure. The owner will most likely be a large multisig. We mentioned the Frax Multisig in the scope too. If moving funds, it is assumed someone in the multisig would catch an invalid or malicious address.

Oxean (judge) commented:

Wardens have demonstrated a mechanism which breaks core assumptions of the contract's accounting. While I am usually very apprehensive to call input sanitization a M issue, a simple require statement here mitigates a risk of accidentally breaking a core tenet of the asset backed token.

Going to award this as Medium for now, may come back to it to revise later.



[M-09] `recoverEther` **not updating** `currentWithheldETH`
breaks calculation of withheld amount for further deposits

The emergency exit function `recoverEther` allows the owner to retrieve the ETH in case an issue were to happen.

The problem is that this function does not update `currentWithheldETH`.

This means upon deposit starting again after the emergency recovery, `currentWithheldETH` will have an offset and will not match the `withholdRatio`.

Direct consequences:

- `depositEther` may not deposit the expected amount of ETH into the ETH 2.0 staking contract.
- The amount of ETH moved to an external yield protocol using `moveWithheldETH()` will be higher than what it should be.



Proof Of Concept

- `frxETHMinter` has a `withholdRatio` set to $2 * 1e5$ - ie the contract is meant to hold 20% of the ETH deposited.
- Users deposit ETH into the contract.
- An issue happens and the owner calls `recoverEther(address(this).balance)`. Before the call, the total balance was $1e20$ (100 ETH), and `currentWithheldETH == 2 * 1e19` - for simplicity we assume no calls to `moveWithheldETH` or `depositEther` have been done yet.
- The ETH balance of the minter is now 0, but `currentWithheldETH` is still $2 * 1e19$
- Users start depositing again.
- At this point, the total balance is now $1e20$ (100 ETH), and `currentWithheldETH == 4 * 1e19`
- The owner calling `depositEther` deposits 32 ether instead of 64 ether, because `currentWithheldETH == 40 ether` instead of 20 ether. The owner

can also call `moveWithheldETH` with `amount == 4 * 1e19` instead of `amount == 2 * 1e19`.

You can add the following Foundry test in `frxETHMinter.t.sol` to reproduce the issue:

- First declare `address Alice = address(1);` before the `setUp()`

```
function testIssueRecoverEther() public {
    vm.startPrank(FRAX_COMPROLLER);

    // Note the starting ETH balance of the comptroller
    uint256 starting_eth = FRAX_COMPROLLER.balance;

    // Give Alice 200 eth
    vm.deal(Alice, 200 ether);
    // Set the withhold ratio to 20% (2 * 1e5)
    minter.setWithholdRatio(200000);
    vm.stopPrank();

    vm.startPrank(Alice);

    //deposit 100 ETH
    minter.submit{ value: 100 ether }();
    vm.stopPrank();

    vm.startPrank(FRAX_COMPROLLER);
    // Recover all
    minter.recoverEther(100 ether);

    // Make sure the FRAX_COMPROLLER got 100 ether back
    assertEq(FRAX_COMPROLLER.balance, starting_eth + (100 ether));

    //check `currentWithheldETH`: it has not been reset and
    assertEq(minter.currentWithheldETH(), 20 ether);
    vm.stopPrank();

    vm.startPrank(Alice);
    //deposit 100 ETH
    minter.submit{ value: 100 ether }();
    //check `currentWithheldETH`: because of the offset, it
    assertEq(minter.currentWithheldETH(), 40 ether);
    assertEq(address(minter).balance, 100 ether);
    vm.stopPrank();
}
```

```

        vm.startPrank(FRAX_COMPROLLER);
        //Owner can call moveWithheldETH, transferring more than
        minter.moveWithheldETH(payable(address(Alice)), 40 ether);
        assertEq(address(minter).balance, 60 ether);
        vm.stopPrank();
    }
}

```



Tools Used

Foundry



Recommended Mitigation Steps

Update `currentWithheldETH` in `recoverEther` :

```

+          currentWithheldETH = currentWithheldETH >= amount ?
192:      (bool success,) = address(owner).call{ value: amount }
193:      require(success, "Invalid transfer");
194:
195:      emit EmergencyEtherRecovered(amount);

```

FortisFortuna (Frax) commented:

@denett

`withholdRatio` is not an iron rule and can be updated by the owner at will. `recoverEther` will likely only be used when we are migrating to a new minting contract, so the accounting in that case is no longer important.

Oxean (judge) commented:

Issue 346 has some great suggestions in it on ensuring user safety in an emergency scenario and think that both of these issues do highlight a valid concern that ultimately could affect the protocol in an emergency scenario.



[M-10] `sfrxETH`: The volatile result of `previewMint()` may prevent `mintWithSignature` from working

Submitted by cccz, also found by rotcivegaf, Trust, and wagmi

In sfrxETH contracts, the result of `previewMint()` changes with the state of the contract, which causes the value of amount to be volatile in the `mintWithSignature` function when `approveMax` is false.

And when using the `mintWithSignature` function, which requires the user to sign for an accurate amount value, when the amount used differs from the result of `previewMint()`, `mintWithSignature` will not work.

Consider the following scenarios.

User A signs using `amount = 1000` and calls the `mintWithSignature` function.

During execution, the previous transaction in the same block changes the state of the contract so that `previewMint(shares) == 1001`, so the transaction is reverted due to a signature check failure.



Proof of Concept

<https://github.com/code-423n4/2022-09-frax/blob/55ea6b1ef3857a277e2f47d42029bc0f3d6f9173/src/sfrxETH.sol#L75-L87>

<https://github.com/transmissions11/solmate/blob/bff24e835192470ed38bf15dbed6084c2d723ace/src/mixins/ERC4626.sol#L140-L144>



Recommended Mitigation Steps

Consider that in the `mintWithSignature` function, the user provides a `maxAmount`, and then requires `maxAmount >= previewMint(shares)` and uses `maxAmount` to verify the signature.

[FortisFortuna \(Frax\) acknowledged and commented:](#)

Technically correct, though in practice, we will allow user-defined slippage on the UI.

[Oxean \(judge\) commented:](#)

I don't believe the UI will be able to assist with this issue unless modifications are made to the smart contracts. The signature will become invalidated due to the return value of `previewMint()` changing while the transaction is waiting to be included in a block.



Low Risk and Non-Critical Issues

For this contest, 83 reports were submitted by wardens detailing low risk and non-critical issues. The [report highlighted below](#) by [rotcivegaf](#) received the top score from the judge.

The following wardens also submitted reports: [Ox1f8b](#), [bytera](#), [OxNazgul](#), [leosathya](#), [gogo](#), [Rolezn](#), [neko_nyaa](#), [lllllll](#), [brgltd](#), [bobirichman](#), [c3phas](#), [CodingNameKiki](#), [ajtra](#), [Ox4non](#), [Deivitto](#), [OxSmartContract](#), [B2](#), [delfin454000](#), [lukris02](#), [Aymen0909](#), [cryptostellar5](#), [rbserver](#), [Bnke0x0](#), [RockingMiles](#), [Diana](#), [Waze](#), [oyc_109](#), [cryptphi](#), [__141345__](#), [mics](#), [tnevler](#), [V_B](#), [aysha](#), [Oxf15ers](#), [a12jmx](#), [Triangle](#), [ayeslick](#), [csanuragjain](#), [Funen](#), [Trust](#), [datapunk](#), [Bahurum](#), [joestakey](#), [8olidity](#), [ladboy233](#), [sikorico](#), [slowmoses](#), [asutorufos](#), [sach1r0](#), [TomJ](#), [Soosh](#), [JLevick](#), [durianSausage](#), [rokinot](#), [JC](#), [bbuddha](#), [yasir](#), [PaludoX0](#), [peritoflores](#), [yongskiws](#), [obront](#), [millersplanet](#), [Lambda](#), [OptimismSec](#), [rvierdiiev](#), [seyni](#), [parashar](#), [Yiko](#), [Tointer](#), [KIntern_NA](#), [Tagir2003](#), [jag](#), [karanctf](#), [exd0tpty](#), [ronnyx2017](#), [natzuu](#), [Ox040](#), [Sm4rty](#), [ret2basic](#), [got_targ](#), [Ch_301](#), and [bharg4v](#).



Low Risk Issues

	Issue	Instances	
L-01	Draft OpenZeppelin Dependencies	1	
L-02	Don't use <code>owner</code> and <code>timelock</code>	2	

Total: 3 instances over 2 issues



[L-01] Draft OpenZeppelin Dependencies

The `ERC20PermitPermissionedMint.sol` contract heredit from an OpenZeppelin contract who is still a draft and is not considered ready for mainnet use.

OpenZeppelin contracts may be considered draft contracts if they have not received adequate security auditing or are liable to change with future development.



Recommendation

Ensure the development team is aware of the risks of using a draft contract or consider waiting until the contract is finalised.

Otherwise, make sure that development team are aware of the risks of using a draft OpenZeppelin contract and accept the risk-benefit trade-off.

Also could evaluate changing to the [solmate contracts](#) since his [ERC20 implementation](#) already has the [EIP-2612 permit](#)

```
File: /src/ERC20/ERC20PermitPermissionedMint.sol
```

```
6 import "openzeppelin-contracts/contracts/token/ERC20/extension
```



[L-02] Don't use `owner` and `timelock`

Using a `timelock` contract gives confidence to the user, but when check

`onlyByOwnGov` allow the `owner` and the `timelock` The `owner` manipulates the contract without a lock time period.



Recommendation

- Use only `Owned` permission
- Remove the `timelock_address`
- The owner should be the `timelock` contract

```
File: /src/frxETH.sol
```

```
38     address _timelock_address
```

```
40     ERC20PermitPermissionedMint(_creator_address, _timelock_ac
```

File: /src/ERC20/ERC20PermitPermissionedMint.sol

```
16     address public timelock_address;

26         address _timelock_address,

34     timelock_address = _timelock_address;

41     require(msg.sender == timelock_address || msg.sender

94     function setTimelock(address _timelock_address) public or
95         require(_timelock_address != address(0), "Zero address
96         timelock_address = _timelock_address;
97         emit TimelockChanged(_timelock_address);
98     }

106     event TimelockChanged(address timelock_address);
```

File: /src/frxETH.sol

```
38     address _timelock_address

40     ERC20PermitPermissionedMint(_creator_address, _timelock_ac
```

File: /src/OperatorRegistry.sol

```
38     address public timelock_address;

40     constructor(address _owner, address _timelock_address, by
41         timelock_address = _timelock_address;

46         require(msg.sender == timelock_address || msg.sender

202     function setTimelock(address _timelock_address) external
203         require(_timelock_address != address(0), "Zero address
204         timelock_address = _timelock_address;
205         emit TimelockChanged(_timelock_address);
206     }

208     event TimelockChanged(address timelock_address);
```



```
File: /src/frxETHMinter.sol
```

```
57         address _timelock_address,
```

```
59     ) OperatorRegistry(_owner, _timelock_address, _withdrawalC
```



Non-Critical Issues

	Issue	Instances
N-01	Unused imports	2
N-02	Non-library/interface files should use fixed compiler versions, not floating ones	6
N-03	Lint	11
N-04	Event is missing <code>indexed</code> fields	19
N-05	Functions, parameters and variables in snake case	31
N-06	Wrong <code>event</code> parameter name	2
N-07	Simplify <code>depositWithSignature</code> function	1
N-08	Missing zero address checks	9

Total: 81 instances over 8 issues



[N-01] Unused imports

```
File: /src/ERC20/ERC20PermitPermissionedMint.sol
```

```
4 import "openzeppelin-contracts/contracts/token/ERC20/ERC20.sol"
```

```
5 import "openzeppelin-contracts/contracts/token/ERC20/IERC20.sc
```



[N-02] Non-library/interface files should use fixed compiler versions, not floating ones

File: /src/ERC20/ERC20PermitPermissionedMint.sol

```
2 pragma solidity ^0.8.0;
```

File: /src/frxETH.sol

```
2 pragma solidity ^0.8.0;
```

File: /src/sfrxETH.sol

```
2 pragma solidity ^0.8.0;
```

File: /src/frxETHMinter.sol

```
2 pragma solidity ^0.8.0;
```

File: /src/OperatorRegistry.sol

```
2 pragma solidity ^0.8.0;
```

File: /src/xERC4626.sol

```
4 pragma solidity ^0.8.0;
```



[N-03] Lint

Wrong indentation:

File: /src/ERC20/ERC20PermitPermissionedMint.sol

```

From:
30     ERC20(_name, _symbol)
31     ERC20Permit(_name)
32     Owned(_creator_address)
To:
30         ERC20(_name, _symbol)
31         ERC20Permit(_name)
32         Owned(_creator_address)

```

File: /src/frxETH.sol

```

From:
37     address _creator_address,
38     address _timelock_address
To:
37         address _creator_address,
38         address _timelock_address

```

```

From:
40     ERC20PermitPermissionedMint(_creator_address, _timelock_ac
To:
40         ERC20PermitPermissionedMint(_creator_address, _timeloc

```

Don't use extra parenthesis:

```

File: /src/sfrxETH.sol

70         return (deposit(assets, receiver));

86         return (mint(shares, receiver));

```

Missed space:

```

File: /src/ERC20/ERC20PermitPermissionedMint.sol

84:56         for (uint i = 0; i < minters_array.length; i++){

```

Remove space:

```
File: /src/ERC20/ERC20PermitPermissionedMint.sol
```

```
63 \n
```

```
File: /src/frxETH.sol
```

```
34 \n
```

```
42 \n
```

```
File: /src/sfrxETH.sol
```

```
88 \n
```

```
File: /src/OperatorRegistry.sol
```

```
29 \n
```

[N-04] 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 (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

```
File: /src/frxETHMinter.sol
```

```
205     event EmergencyEtherRecovered(uint256 amount);
```

```
206     event EmergencyERC20Recovered(address tokenAddress, uint2
```

```
207     event ETHSubmitted(address indexed sender, address indexe
```

```
208     event DepositEtherPaused(bool new_status);
```

```

209     event DepositSent(bytes indexed pubKey, bytes withdrawalC

210     event SubmitPaused(bool new_status);

211     event WithheldETHMoved(address indexed to, uint256 amount

212     event WithholdRatioSet(uint256 newRatio);

```

File: /src/OperatorRegistry.sol

```

208     event TimelockChanged(address timelock_address);

209     event WithdrawalCredentialSet(bytes _withdrawalCredential

210     event ValidatorAdded(bytes pubKey, bytes withdrawalCreder

212     event ValidatorRemoved(bytes pubKey, uint256 remove_idx,

213     event ValidatorsPopped(uint256 times);

214     event ValidatorsSwapped(bytes from_pubKey, bytes to_pubKe

```

File: /src/ERC20/ERC20PermitPermissionedMint.sol

```

102     event TokenMinterBurned(address indexed from, address inc

103     event TokenMinterMinted(address indexed from, address inc

104     event MinterAdded(address minter_address);

105     event MinterRemoved(address minter_address);

106     event TimelockChanged(address timelock_address);

```



[N-05] Functions, parameters and variables in snake case

Use camel case for all functions, parameters and variables and snake case for constants

File: /src/ERC20/ERC20PermitPermissionedMint.sol

```

16     address public timelock_address;

19     address[] public minters_array; // Allowed to mint

25         address _creator_address,

26         address _timelock_address,

53     function minter_burn_from(address b_address, uint256 b_an

59     function minter_mint(address m_address, uint256 m_amount)

65     function addMinter(address minter_address) public onlyByC

76     function removeMinter(address minter_address) public only

94     function setTimelock(address _timelock_address) public or

104    event MinterAdded(address minter_address);

105    event MinterRemoved(address minter_address);

106    event TimelockChanged(address timelock_address);

```

File: /src/frxETH.sol

```

37         address _creator_address,

38         address _timelock_address

```

File: /src/frxETHMinter.sol

```

57         address _timelock_address,

78         uint256 sfrxeth_recieved = sfrxETHToken.deposit(msg.v

94         uint256 withheld_amt = 0;

208    event DepositEtherPaused(bool new_status);

210    event SubmitPaused(bool new_status);

```

File: /src/OperatorRegistry.sol

```
37     bytes curr_withdrawal_pubkey; // Pubkey for ETH 2.0 withd

38     address public timelock_address;

40     constructor(address _owner, address _timelock_address, by

69     function swapValidator(uint256 from_idx, uint256 to_idx)

93     function removeValidator(uint256 remove_idx, bool dont_ca

95         bytes memory removed_pubkey = validators[remove_idx].

108         Validator[] memory original_validators = validate

181     function setWithdrawalCredential(bytes memory _new_withdr

202     function setTimelock(address _timelock_address) external

208     event TimelockChanged(address timelock_address);

212     event ValidatorRemoved(bytes pubKey, uint256 remove_idx,

214     event ValidatorsSwapped(bytes from_pubKey, bytes to_pubKe
```

[N-06] Wrong event parameter name

Replace to parameter of TokenMinterBurned event to minter Replace from parameter of TokenMinterMinted event to minter

File: /src/ERC20/ERC20PermitPermissionedMint.sol

```
102     event TokenMinterBurned(address indexed from, address inc

103     event TokenMinterMinted(address indexed from, address inc
```

[N-07] Simplify depositWithSignature function

The parameter approveMax of depositWithSignature function could be removedready, the permit assets should be always equal to deposit assets

```
/// @notice Approve and deposit() in one transaction
function depositWithSignature(
    uint256 assets,
    address receiver,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external nonReentrant returns (uint256 shares) {
    asset.permit(msg.sender, address(this), assets, deadline
    return (deposit(assets, receiver));
}
```



[N-08] Missing zero address checks

File: /src/ERC20/ERC20PermitPermissionedMint.sol

```
26         address _timelock_address,
```

File: /src/sfrxETH.sol

```
42     constructor(ERC20 _underlying, uint32 _rewardsCycleLength)
```

File: /src/frxETHMinter.sol

```
53         address depositContractAddress,
```

```
54         address frxETHAddress,
```

```
55         address sfrxETHAddress,
```

```
57         address _timelock_address,
```

```
70     function submitAndDeposit(address recipient) external pay
```

```
166     function moveWithheldETH(address payable to, uint256 amo
```



```
File: /src/OperatorRegistry.sol
```

```
/*_timelock_address parameter*/  
40     constructor(address _owner, address _timelock_address, by
```



Gas Optimizations

For this contest, 93 reports were submitted by wardens detailing gas optimizations. The [report highlighted below](#) by pfapostol received the top score from the judge.

The following wardens also submitted reports: [lllllll](#), [ReyAdmirado](#), [ajtra](#), [OxSmartContract](#), [JC](#), [Rolezn](#), [rotcivegaf](#), [c3phas](#), [oyc_109](#), [Bnke0x0](#), [__141345__](#), [TomJ](#), [ret2basic](#), [Sm4rty](#), [prasantgupta52](#), [Aymen0909](#), [Diana](#), [cryptostellar5](#), [SnowMan](#), [ch0bu](#), [gogo](#), [B2](#), [peanuts](#), [Deivitto](#), [medikko](#), [0x1f8b](#), [zishansami](#), [rbserver](#), [Rohan16](#), [erictree](#), [durianSausage](#), [d3e4](#), [OxNazgul](#), [RockingMiles](#), [karancf](#), [RaymondFam](#), [0xA5DF](#), [brgltd](#), [natzuu](#), [0x040](#), [lukris02](#), [tnevler](#), [got_targ](#), [Tomio](#), [Amithuddar](#), [Metatron](#), [samruna](#), [millersplanet](#), [drdr](#), [leosathya](#), [Waze](#), [bulej93](#), [jag](#), [Satyam_Sharma](#), [slowmoses](#), [ronnyx2017](#), [Ocean_Sky](#), [imare](#), [JAGADESH](#), [SooYa](#), [V_B](#), [Pheonix](#), [neko_nyaa](#), [sach1r0](#), [delfin454000](#), [0x4non](#), [Fitraldys](#), [aysha](#), [Oxsam](#), [ladboy233](#), [0x5rings](#), [fatherOfBlocks](#), [Triangle](#), [seyni](#), [albincsergo](#), [Tagir2003](#), [bytera](#), [beardofginger](#), [PaludoX0](#), [Ben](#), [Chom](#), [rokinot](#), [Funen](#), [CodingNameKiki](#), [asutorufos](#), [emrekocak](#), [wagmi](#), [dharma09](#), [Oxmatt](#), [mics](#), [bobirichman](#), and [cryptphi](#).



Gas Optimizations Summary

Gas savings are estimated using the gas report of existing `FORGE_GAS_REPORT=true`
`forge test --fork-url https://eth-mainnet.g.alchemy.com/v2/<API> tests`
(the sum of all deployment costs and the sum of the costs of calling methods) and may vary depending on the implementation of the fix.

Note: method call evaluations are volatile: $\approx \pm 500$

	Issue	Inst an ces	Estimate d gas(depl oyments)	Estimated gas(avg method call)	Estimated gas(min method call)	Estimated gas(max method call)
G-01	Deleting an array element can use a more efficient algorithm	1	23 830	271 820	5 298	538 343
G-02	Use function instead of modifiers	4	177 805	-990	-389	1 902
G-03	Use custom errors rather than revert()/require() strings to save gas	21	150 574	-123	-25	-184
G-04	Using bools for storage incurs overhead	3	20 221	-990	266	-5 979
G-05	Unchecking arithmetics operations that can't underflow/overflow	7	18 621	503	227	829
G-06	<code>storage</code> pointer to a structure is cheaper than copying each value of the structure into <code>memory</code> , same for <code>array</code> and <code>mapping</code>	1	8 208	-970	106	2 487
G-07	<code>x = x + y</code> is more efficient, than <code>x += y</code>	4	5 007	87	82	101
G-08	It costs more gas to initialize non-constant/non-immutable variables to zero than to let the default of zero be applied	2	4 415	0	0	0
G-09	Don't compare boolean expressions to boolean literals	3	3 006	-477	43	55
G-10	State variables should be cached in stack variables rather than re-reading them from storage	1	400	511	-21	4 839
	Overall Gas Savings	47	419 688(7,43%)	270 705(12,18%)	5 474(0,42%)	539 594(18,02%)

Total: 47 instances over 10 issues

[G-01] Deleting an array element can use a more efficient algorithm (1 instance)

- Deployment. Gas Saved: **23 830**
- Minumal Method Call. Gas Saved: **5 298**
- Average Method Call. Gas Saved: **271 820**
- Maximum Method Call. Gas Saved: **538 343**



src/OperatorRegistry.sol:107-116

```
diff --git a/src/OperatorRegistry.sol b/src/OperatorRegistry.sol
index f81094c..6732da9 100644
--- a/src/OperatorRegistry.sol
+++ b/src/OperatorRegistry.sol
@@ -104,18 +104,13 @@ contract OperatorRegistry is Owned {
     104, 104:         }
     105, 105:         // More gassy, loop
     106, 106:         else {
- 107      :-         // Save the original validators
- 108      :-         Validator[] memory original_validators =
- 109      :-
- 110      :-         // Clear the original validators list
- 111      :-         delete validators;
- 112      :-
- 113      :-         // Fill the new validators array with al
- 114      :-         for (uint256 i = 0; i < original_validat
- 115      :-             if (i != remove_idx) {
- 116      :-                 validators.push(original_validat
+ 107:++         uint256 length = validators.length - 1;
+ 108:++         unchecked {
+ 109:++             for (uint256 i = remove_idx; i < ler
+ 110:++                 validators[i] = validators[i + 1
     117, 111:         }
     118, 112:         }
+ 113:++         validators.pop();
     119, 114:         }
     120, 115:
     121, 116:         emit ValidatorRemoved(removed_pubkey, remove
```



[G-02] Use function instead of modifiers (4 instances)

- Deployment. Gas Saved: **177 805**
- Minumal Method Call. Gas Saved: **-389**
- Average Method Call. Gas Saved: **-990**
- Maximum Method Call. Gas Saved: **1 902**



src/ERC20/ERC20PermitPermissionedMint.sol:40, 45

```
diff --git a/src/ERC20/ERC20PermitPermissionedMint.sol b/src/ERC20/ERC20PermitPermissionedMint.sol
index 3bed26d..78da7f1 100644
--- a/src/ERC20/ERC20PermitPermissionedMint.sol
+++ b/src/ERC20/ERC20PermitPermissionedMint.sol
@@ -37,32 +37,33 @@ contract ERC20PermitPermissionedMint is ERC20
     37, 37:
     38, 38:         /* ===== MODIFIERS ===== */
     39, 39:
-   40     :-     modifier onlyByOwnGov() {
+   40     40:+     function onlyByOwnGov() private {
     41, 41:         require(msg.sender == timelock_address || ms
-   42     :-         _;
+   42     42:     }
     43, 42:     }
     44, 43:
-   45     :-     modifier onlyMinters() {
+   44     44:+     function onlyMinters() private {
     46, 45:         require(minters[msg.sender] == true, "Only mi
-   47     :-         _;
+   46     46:     }
     48, 46:     }
     49, 47:
     50, 48:         /* ===== RESTRICTED FUNCTIONS ===== */
     51, 49:
     52, 50:         // Used by minters when user redeems
-   53     :-     function minter_burn_from(address b_address, uir
+   51     51:+     function minter_burn_from(address b_address, uir
+   52     52:+         onlyMinters();
     54, 53:         super.burnFrom(b_address, b_amount);
     55, 54:         emit TokenMinterBurned(b_address, msg.sender
     56, 55:     }
     57, 56:
     58, 57:         // This function is what other minters will call
-   59     :-     function minter_mint(address m_address, uint256
+   58     58:+     function minter_mint(address m_address, uint256
+   59     59:+         onlyMinters();
```

```

60, 60:         super._mint(m_address, m_amount);
61, 61:         emit TokenMinterMinted(msg.sender, m_address);
62, 62:     }
63, 63:
64, 64:     // Adds whitelisted minters
- 65     :-     function addMinter(address minter_address) public
+ 65: +     function addMinter(address minter_address) public
+ 66: +         onlyByOwnGov();
66, 67:         require(minter_address != address(0), "Zero
67, 68:
68, 69:         require(minters[minter_address] == false, "Z
@@ -73,7 +74,8 @@ contract ERC20PermitPermissionedMint is ERC20F
73, 74:     }
74, 75:
75, 76:     // Remove a minter
- 76     :-     function removeMinter(address minter_address) pu
+ 77: +     function removeMinter(address minter_address) pu
+ 78: +         onlyByOwnGov();
77, 79:         require(minter_address != address(0), "Zero
78, 80:         require(minters[minter_address] == true, "Ac
79, 81:
@@ -91,7 +93,8 @@ contract ERC20PermitPermissionedMint is ERC20F
91, 93:         emit MinterRemoved(minter_address);
92, 94:     }
93, 95:
- 94     :-     function setTimelock(address _timelock_address)
+ 96: +     function setTimelock(address _timelock_address)
+ 97: +         onlyByOwnGov();
95, 98:         require(_timelock_address != address(0), "Ze
96, 99:         timelock_address = _timelock_address;
97, 100:         emit TimelockChanged(_timelock_address);

```



src/OperatorRegistry.sol:45

```

diff --git a/src/OperatorRegistry.sol b/src/OperatorRegistry.sol
index f81094c..fc5d16d 100644
--- a/src/OperatorRegistry.sol
+++ b/src/OperatorRegistry.sol
@@ -42,15 +42,15 @@ contract OperatorRegistry is Owned {
    42, 42:         curr_withdrawal_pubkey = _withdrawal_pubkey;
    43, 43:     }
    44, 44:
- 45     :-     modifier onlyByOwnGov() {

```

```

+      45: function onlyByOwnGov() internal {
+      46, 46:         require(msg.sender == timelock_address || ms
-      47      :-         _;
+      48, 47:     }
+      49, 48:
+      50, 49:     /// @notice Add a new validator
+      51, 50:     /** @dev You should verify offchain that the val
+      52, 51:         Reason we don't do that here is for gas */
-      53      :-     function addValidator(Validator calldata validat
+      52: function addValidator(Validator calldata validat
+      53:         onlyByOwnGov());
+      54, 54:         validators.push(validator);
+      55, 55:         emit ValidatorAdded(validator.pubKey, curr_v
+      56, 56:     }
@@ -58,7 +58,8 @@ contract OperatorRegistry is Owned {
+      58, 58:     /// @notice Add multiple new validators in one f
+      59, 59:     /** @dev You should verify offchain that the val
+      60, 60:         Reason we don't do that here is for gas */
-      61      :-     function addValidators(Validator[] calldata vali
+      61: function addValidators(Validator[] calldata vali
+      62:         onlyByOwnGov());
+      62, 63:         uint arrayLength = validatorArray.length;
+      63, 64:         for (uint256 i = 0; i < arrayLength; ++i) {
+      64, 65:             addValidator(validatorArray[i]);
@@ -66,7 +67,8 @@ contract OperatorRegistry is Owned {
+      66, 67:     }
+      67, 68:
+      68, 69:     /// @notice Swap the location of one validator v
-      69      :-     function swapValidator(uint256 from_idx, uint256
+      70: function swapValidator(uint256 from_idx, uint256
+      71:         onlyByOwnGov());
+      70, 72:         // Get the original values
+      71, 73:         Validator memory fromVal = validators[from_i
+      72, 74:         Validator memory toVal = validators[to_idx];
@@ -79,7 +81,8 @@ contract OperatorRegistry is Owned {
+      79, 81:     }
+      80, 82:
+      81, 83:     /// @notice Remove validators from the end of th
-      82      :-     function popValidators(uint256 times) public onl
+      84: function popValidators(uint256 times) public {
+      85:         onlyByOwnGov();
+      83, 86:         // Loop through and remove validator entries
+      84, 87:         for (uint256 i = 0; i < times; ++i) {
+      85, 88:             validators.pop();
@@ -90,7 +93,8 @@ contract OperatorRegistry is Owned {
+      90, 93:

```

```

91, 94:      /** @notice Remove a validator from the array. ]
92, 95:      a swap and pop will occur instead of a more
- 93      :-      function removeValidator(uint256 remove_idx, boc
+          96:+      function removeValidator(uint256 remove_idx, boc
+          97:+      onlyByOwnGov();
94, 98:      // Get the pubkey for the validator to remov
95, 99:      bytes memory removed_pubkey = validators[ren
96, 100:
@@ -178,7 +182,8 @@ contract OperatorRegistry is Owned {
178, 182:
179, 183:      /// @notice Requires empty validator stack as ch
180, 184:      /// @dev May need to call clearValidatorArray()
- 181      :-      function setWithdrawalCredential(bytes memory _r
+          185:+      function setWithdrawalCredential(bytes memory _r
+          186:+      onlyByOwnGov();
182, 187:      require(numValidators() == 0, "Clear validat
183, 188:      curr_withdrawal_pubkey = _new_withdrawal_puk
184, 189:
@@ -187,7 +192,8 @@ contract OperatorRegistry is Owned {
187, 192:
188, 193:      /// @notice Empties the validator array
189, 194:      /// @dev Need to do this before setWithdrawalCre
- 190      :-      function clearValidatorArray() external onlyByOv
+          195:+      function clearValidatorArray() external {
+          196:+      onlyByOwnGov();
191, 197:      delete validators;
192, 198:
193, 199:      emit ValidatorArrayCleared();
@@ -199,7 +205,8 @@ contract OperatorRegistry is Owned {
199, 205:      }
200, 206:
201, 207:      /// @notice Set the timelock contract
- 202      :-      function setTimelock(address _timelock_address)
+          208:+      function setTimelock(address _timelock_address)
+          209:+      onlyByOwnGov();
203, 210:      require(_timelock_address != address(0), "Ze
204, 211:      timelock_address = _timelock_address;
205, 212:      emit TimelockChanged(_timelock_address);

```



[src/frxETHMinter.sol:link](#)

```

diff --git a/src/frxETHMinter.sol b/src/frxETHMinter.sol
index 4565883..2690157 100644

```

```

--- a/src/frxETHMinter.sol
+++ b/src/frxETHMinter.sol
@@ -156,14 +156,16 @@ contract frxETHMinter is OperatorRegistry,
    156, 156:
    157, 157:      /// @param newRatio of ETH that is sent to depos
    158, 158:      /// @notice An input of 1e6 results in 100% of E
- 159      :-      function setWithholdRatio(uint256 newRatio) exte
+ 159      :-      function setWithholdRatio(uint256 newRatio) exte
+ 160      :-      function setWithholdRatio(uint256 newRatio) exte
+ 160      :-      onlyByOwnGov();
    160, 161:      require (newRatio <= RATIO_PRECISION, "Ratic
    161, 162:      withholdRatio = newRatio;
    162, 163:      emit WithholdRatioSet(newRatio);
    163, 164:      }
    164, 165:
    165, 166:      /// @notice Give the withheld ETH to the "to" ac
- 166      :-      function moveWithheldETH(address payable to, uir
+ 167      :-      function moveWithheldETH(address payable to, uir
+ 168      :-      function moveWithheldETH(address payable to, uir
+ 168      :-      onlyByOwnGov();
    167, 169:      require(amount <= currentWithheldETH, "Not e
    168, 170:      currentWithheldETH -= amount;
    169, 171:
@@ -174,21 +176,24 @@ contract frxETHMinter is OperatorRegistry,
    174, 176:      }
    175, 177:
    176, 178:      /// @notice Toggle allowing submites
- 177      :-      function togglePauseSubmits() external onlyByOwr
+ 179      :-      function togglePauseSubmits() external {
+ 180      :-      onlyByOwnGov();
    178, 181:      submitPaused = !submitPaused;
    179, 182:
    180, 183:      emit SubmitPaused(submitPaused);
    181, 184:      }
    182, 185:
    183, 186:      /// @notice Toggle allowing depositing ETH to va
- 184      :-      function togglePauseDepositEther() external only
+ 187      :-      function togglePauseDepositEther() external {
+ 188      :-      onlyByOwnGov();
    185, 189:      depositEtherPaused = !depositEtherPaused;
    186, 190:
    187, 191:      emit DepositEtherPaused(depositEtherPaused);
    188, 192:      }
    189, 193:
    190, 194:      /// @notice For emergencies if something gets st
- 191      :-      function recoverEther(uint256 amount) external c
+ 195      :-      function recoverEther(uint256 amount) external {
+ 196      :-      onlyByOwnGov();

```



```

192, 197:      (bool success,) = address(owner).call{ value
193, 198:      require(success, "Invalid transfer");
194, 199:
@@ -196,7 +201,8 @@ contract frxETHMinter is OperatorRegistry, F
196, 201:      }
197, 202:
198, 203:      /// @notice For emergencies if someone accidenta
- 199      :-      function recoverERC20(address tokenAddress, uint
+      204:+      function recoverERC20(address tokenAddress, uint
+      205:+          onlyByOwnGov();
200, 206:      require(IERC20(tokenAddress).transfer(owner,
201, 207:
202, 208:      emit EmergencyERC20Recovered(tokenAddress, t

```



[G-03] Use custom errors rather than revert()/require() strings to save gas (21 instances)

- Deployment. Gas Saved: **150 574**
- Minumal Method Call. Gas Saved: **-25**
- Average Method Call. Gas Saved: **-123**
- Maximum Method Call. Gas Saved: **-184**

Custom errors are available from solidity version 0.8.4. Custom errors save ~50 gas each time they're hit by [avoiding having to allocate and store the revert string](#). Not defining the strings also save deployment gas



src/ERC20/ERC20PermitPermissionedMint.sol:41, 46, 66, 68, 77-78, 95

```

diff --git a/src/ERC20/ERC20PermitPermissionedMint.sol b/src/ERC
index 3bed26d..758ca2a 100644
--- a/src/ERC20/ERC20PermitPermissionedMint.sol
+++ b/src/ERC20/ERC20PermitPermissionedMint.sol
@@ -7,6 +7,13 @@ import "openzeppelin-contracts/contracts/token/
7, 7: import "openzeppelin-contracts/contracts/token/ERC20
8, 8: import "../Utils/Owned.sol";
9, 9:
+      10:+
+      11:+error ZeroAddressDecteded();
+      12:+error AddresssNonExists();

```

```

+      13:+error AddressAlreadyExists();
+      14:+error OnlyMinters();
+      15:+error NotOwnerOrTimelock();
+      16:+
10,    17: /// @title Parent contract for frxETH.sol
11,    18: /** @notice Combines Openzeppelin's ERC20Permit and
12,    19:     Also includes a list of authorized minters */
@@ -38,12 +45,12 @@ contract ERC20PermitPermissionedMint is ERC2
38,    45:     /* ===== MODIFIERS ===== */
39,    46:
40,    47:     modifier onlyByOwnGov() {
- 41      :-         require(msg.sender == timelock_address || ms
+      48:+         if(msg.sender != timelock_address && msg.ser
42,    49:         _;
43,    50:     }
44,    51:
45,    52:     modifier onlyMinters() {
- 46      :-         require(minters[msg.sender] == true, "Only mi
+      53:+         if(minters[msg.sender] != true) revert OnlyMi
47,    54:         _;
48,    55:     }
49,    56:
@@ -63,9 +70,10 @@ contract ERC20PermitPermissionedMint is ERC20
63,    70:
64,    71:     // Adds whitelisted minters
65,    72:     function addMinter(address minter_address) publi
- 66      :-         require(minter_address != address(0), "Zero
+      73:+         if(minter_address == address(0)) revert Zero
67,    74:
- 68      :-         require(minters[minter_address] == false, "Z
+      75:+
+      76:+         if(minters[minter_address] != false) revert
69,    77:         minters[minter_address] = true;
70,    78:         minters_array.push(minter_address);
71,    79:
@@ -74,8 +82,8 @@ contract ERC20PermitPermissionedMint is ERC20P
74,    82:
75,    83:     // Remove a minter
76,    84:     function removeMinter(address minter_address) pu
- 77      :-         require(minter_address != address(0), "Zero
- 78      :-         require(minters[minter_address] == true, "Ac
+      85:+         if(minter_address == address(0)) revert Zero
+      86:+         if(minters[minter_address] != true) revert Z
79,    87:
80,    88:         // Delete from the mapping
81,    89:         delete minters[minter_address];

```

```

@@ -92,7 +100,7 @@ contract ERC20PermitPermissionedMint is ERC20
    92, 100:    }
    93, 101:
    94, 102:    function setTimelock(address _timelock_address)
-   95        :-        require(_timelock_address != address(0), "Zero
+   103:+        if(_timelock_address == address(0)) revert 2
    96, 104:        timelock_address = _timelock_address;
    97, 105:        emit TimelockChanged(_timelock_address);
    98, 106:    }

```



src/OperatorRegistry.sol:46, 137, 182, 203

```

diff --git a/src/OperatorRegistry.sol b/src/OperatorRegistry.sol
index f81094c..ac3b7a1 100644
--- a/src/OperatorRegistry.sol
+++ b/src/OperatorRegistry.sol
@@ -23,6 +23,12 @@ pragma solidity ^0.8.0;
    23, 23:
    24, 24: import "../Utils/Owned.sol";
    25, 25:
+   26:+error NotOwnerOrTimelock();
+   27:+error ClearValidatorArrayFirst();
+   28:+error ZeroAddressDected();
+   29:+error ValidatorStackEmpty();
+   30:+
+   31:+
    26, 32: /// @title Keeps track of validators used for ETH 2.
    27, 33: /// @notice A permissioned owner can add and remove
    28, 34: contract OperatorRegistry is Owned {
@@ -43,7 +49,7 @@ contract OperatorRegistry is Owned {
    43, 49:    }
    44, 50:
    45, 51:    modifier onlyByOwnGov() {
-   46        :-        require(msg.sender == timelock_address || ms
+   52:+        if(msg.sender != timelock_address && msg.ser
    47, 53:        _;
    48, 54:    }
    49, 55:
@@ -134,7 +140,7 @@ contract OperatorRegistry is Owned {
    134, 140:    {
    135, 141:        // Make sure there are free validators avail
    136, 142:        uint numVals = numValidators();
-   137        :-        require(numVals != 0, "Validator stack is en

```

```

+      143:+      if(numVals == 0) revert ValidatorStackEmpty()
138, 144:
139, 145:      // Pop the last validator off the array
140, 146:      Validator memory popped = validators[numVals]
@@ -179,7 +185,7 @@ contract OperatorRegistry is Owned {
179, 185:      /// @notice Requires empty validator stack as check
180, 186:      /// @dev May need to call clearValidatorArray()
181, 187:      function setWithdrawalCredential(bytes memory _r
- 182      :-      require(numValidators() == 0, "Clear validator
+      188:+      if(numValidators() != 0) revert ClearValidator()
183, 189:      curr_withdrawal_pubkey = _new_withdrawal_pubkey
184, 190:
185, 191:      emit WithdrawalCredentialSet(_new_withdrawal
@@ -200,7 +206,7 @@ contract OperatorRegistry is Owned {
200, 206:
201, 207:      /// @notice Set the timelock contract
202, 208:      function setTimelock(address _timelock_address)
- 203      :-      require(_timelock_address != address(0), "Zero
+      209:+      if(_timelock_address == address(0)) revert ZeroAddress
204, 210:      timelock_address = _timelock_address;
205, 211:      emit TimelockChanged(_timelock_address);
206, 212:      }

```



src/frxETHMinter.sol:79, 87-88, 122, 126, 140, 167, 171, 193, 200

```

diff --git a/src/frxETHMinter.sol b/src/frxETHMinter.sol
index 4565883..f3b5abe 100644
--- a/src/frxETHMinter.sol
+++ b/src/frxETHMinter.sol
@@ -29,6 +29,17 @@ import "openzeppelin-contracts/contracts/token/ERC20/IERC20.sol";
29, 29: import { IDepositContract } from "../DepositContract.sol";
30, 30: import "../OperatorRegistry.sol";
31, 31:
+      32:+error InvalidTransferERC20();
+      33:+error InvalidTransfer();
+      34:+error NotEnoughWithgeld();
+      35:+error AlreadyDeposited();
+      36:+error NotEnoughETH();
+      37:+error DepositPaused();
+      38:+error CannotSubmitZero();
+      39:+error NoSfrxETHReturned();
+      40:+error SubmitIsPaused();
+      41:+

```

```

+      42: +
32, 43: /// @title Authorized minter contract for frxETH
33, 44: /// @notice Accepts user-supplied ETH and converts it
34, 45: /** @dev Has permission to mint frxETH.
@@ -76,7 +87,7 @@ contract frxETHMinter is OperatorRegistry, Recipient {
76, 87:
77, 88:         // Deposit the frxETH and give the generated frxETH token
78, 89:         uint256 sfrxeth_recieved = sfrxETHToken.deposit(msg.value);
- 79      :-         require(sfrxeth_recieved > 0, 'No sfrxETH was deposited');
+      90: +         if(sfrxeth_recieved == 0) revert NoSfrxETHDeposited();
80, 91:
81, 92:         return sfrxeth_recieved;
82, 93:     }
@@ -84,8 +95,8 @@ contract frxETHMinter is OperatorRegistry, Recipient {
84, 95:         /// @notice Mint frxETH to the recipient using the provided
85, 96:         function _submit(address recipient) internal nonReentrant {
86, 97:             // Initial pause and value checks
- 87      :-         require(!submitPaused, "Submit is paused");
- 88      :-         require(msg.value != 0, "Cannot submit 0");
+      98: +         if(submitPaused) revert SubmitIsPaused();
+      99: +         if(msg.value == 0) revert CannotSubmitZero();
89, 100:
90, 101:             // Give the sender frxETH
91, 102:             sfrxETHToken.minter_mint(recipient, msg.value);
@@ -119,11 +130,11 @@ contract frxETHMinter is OperatorRegistry, Recipient {
119, 130:         /// @dev Usually a bot will call this periodically to
120, 131:         function depositEther() external nonReentrant {
121, 132:             // Initial pause check
- 122      :-         require(!depositEtherPaused, "Depositing ETH is paused");
+      133: +         if(depositEtherPaused) revert DepositPaused();
123, 134:
124, 135:             // See how many deposits can be made. Truncate to
125, 136:             uint256 numDeposits = (address(this).balance / msg.value);
- 126      :-         require(numDeposits > 0, "Not enough ETH in account");
+      137: +         if(numDeposits == 0) revert NotEnoughETH();
127, 138:
128, 139:             // Give each deposit chunk to an empty validator
129, 140:             for (uint256 i = 0; i < numDeposits; ++i) {
@@ -137,7 +148,7 @@ contract frxETHMinter is OperatorRegistry, Recipient {
137, 148:
138, 149:             // Make sure the validator hasn't been closed yet
139, 150:             // until withdrawals are allowed
- 140      :-         require(!activeValidators[pubKey], "Validator is closed");
+      151: +         if(activeValidators[pubKey]) revert AlreadyValidated();
141, 152:
142, 153:             // Deposit the ether in the ETH 2.0 deposit contract

```

```

143, 154:         depositContract.deposit{value: DEPOSIT_S
@@ -164,11 +175,11 @@ contract frxETHMinter is OperatorRegistry,
164, 175:
165, 176:         /// @notice Give the withheld ETH to the "to" ac
166, 177:         function moveWithheldETH(address payable to, uir
- 167         :-         require(amount <= currentWithheldETH, "Not e
+         178:+         if(amount > currentWithheldETH) revert NotEr
168, 179:         currentWithheldETH -= amount;
169, 180:
170, 181:         (bool success,) = payable(to).call{ value: a
- 171         :-         require(success, "Invalid transfer");
+         182:+         if(!success) revert InvalidTransfer();
172, 183:
173, 184:         emit WithheldETHMoved(to, amount);
174, 185:     }
@@ -190,14 +201,14 @@ contract frxETHMinter is OperatorRegistry,
190, 201:         /// @notice For emergencies if something gets st
191, 202:         function recoverEther(uint256 amount) external c
192, 203:         (bool success,) = address(owner).call{ value
- 193         :-         require(success, "Invalid transfer");
+         204:+         if(!success) revert InvalidTransfer();
194, 205:
195, 206:         emit EmergencyEtherRecovered(amount);
196, 207:     }
197, 208:
198, 209:         /// @notice For emergencies if someone accidenta
199, 210:         function recoverERC20(address tokenAddress, uint
- 200         :-         require(IERC20(tokenAddress).transfer(owner,
+         211:+         if(!IERC20(tokenAddress).transfer(owner, tok
201, 212:
202, 213:         emit EmergencyERC20Recovered(tokenAddress, t
203, 214:     }

```



test/frxETHMinter.t.sol

```

diff --git a/test/frxETHMinter.t.sol b/test/frxETHMinter.t.sol
index f4d6265..9529428 100644
--- a/test/frxETHMinter.t.sol
+++ b/test/frxETHMinter.t.sol
@@ -3,7 +3,7 @@ pragma solidity ^0.8.0;
3, 3:
4, 4: import { Test } from "forge-std/Test.sol";
5, 5: import { DepositContract } from "../src/DepositContr

```

```

- 6      :-import { frxETHMinter, OperatorRegistry } from "../s
+      6:+import { frxETHMinter, OperatorRegistry, NotEnoughET
      7, 7: import { frxETH } from "../src/frxETH.sol";
      8, 8: import { sfrxETH, ERC20 } from "../src/sfrxETH.sol";
      9, 9:
@@ -223,7 +223,7 @@ contract frxETHMinterTest is Test {
      223, 223:
      224, 224:          // Try having the validator deposit.
      225, 225:          // Should fail due to lack of ETH
- 226      :-          vm.expectRevert("Not enough ETH in contract'
+      226:+          vm.expectRevert(NotEnoughETH.selector);
      227, 227:          minter.depositEther();
      228, 228:
      229, 229:          // Deposit last 1 ETH for frxETH, making the
@@ -239,7 +239,7 @@ contract frxETHMinterTest is Test {
      239, 239:
      240, 240:          // Try having the validator deposit another
      241, 241:          // Should fail due to lack of ETH
- 242      :-          vm.expectRevert("Not enough ETH in contract'
+      242:+          vm.expectRevert(NotEnoughETH.selector);
      243, 243:          minter.depositEther();
      244, 244:
      245, 245:          // Deposit 32 ETH for frxETH
@@ -247,14 +247,14 @@ contract frxETHMinterTest is Test {
      247, 247:
      248, 248:          // Try having the validator deposit another
      249, 249:          // Should fail due to lack of a free validat
- 250      :-          vm.expectRevert("Validator stack is empty");
+      250:+          vm.expectRevert(ValidatorStackEmpty.selector);
      251, 251:          minter.depositEther();
      252, 252:
      253, 253:          // Pause submits
      254, 254:          minter.togglePauseSubmits();
      255, 255:
      256, 256:          // Try submitting while paused (should fail)
- 257      :-          vm.expectRevert("Submit is paused");
+      257:+          vm.expectRevert(SubmitIsPaused.selector);
      258, 258:          minter.submit{ value: 1 ether }();
      259, 259:
      260, 260:          // Unpause submits
@@ -264,7 +264,7 @@ contract frxETHMinterTest is Test {
      264, 264:          minter.togglePauseDepositEther();
      265, 265:
      266, 266:          // Try submitting while paused (should fail)
- 267      :-          vm.expectRevert("Depositing ETH is paused");
+      267:+          vm.expectRevert(DepositPaused.selector);

```

```

268, 268:          minter.depositEther();
269, 269:
270, 270:          // Unpause validator ETH deposits
@@ -303,7 +303,7 @@ contract frxETHMinterTest is Test {
303, 303:
304, 304:          // Try having the validator deposit.
305, 305:          // Should fail due to lack of ETH because ha
- 306      :-      vm.expectRevert("Not enough ETH in contract'
+      306:+      vm.expectRevert(NotEnoughETH.selector);
307, 307:          minter.depositEther();
308, 308:
309, 309:          // Deposit another 32 ETH for frxETH.

```



test/frxETHsfrxETHcombo.t.sol

```

diff --git a/test/frxETH_sfrxETH_combo.t.sol b/test/frxETH_sfrxETH_combo.t.sol
index 5fd1612..be1236c 100644
--- a/test/frxETH_sfrxETH_combo.t.sol
+++ b/test/frxETH_sfrxETH_combo.t.sol
@@ -5,7 +5,7 @@ pragma solidity ^0.8.0;
5, 5: import { Test } from "forge-std/Test.sol";
6, 6: import { frxETH } from "../src/frxETH.sol";
7, 7: import { sfrxETH, ERC20 } from "../src/sfrxETH.sol";
- 8      :-import { frxETHMinter } from "../src/frxETHMinter.sol";
+      8:+import { frxETHMinter, NotEnoughETH, CannotSubmitZero } from "../src/frxETHMinter.sol";
9, 9: import { SigUtils } from "../src/Utils/SigUtils.sol";
10, 10:
11, 11: contract xERC4626Test is Test {
@@ -822,7 +822,7 @@ contract xERC4626Test is Test {
822, 822:          if (transfer_amount > 0) require(owner.balancerContract != address(0));
823, 823:
824, 824:          vm.prank(owner);
- 825      :-      if (transfer_amount == 0) vm.expectRevert("Cannot submit zero");
+      825:+      if (transfer_amount == 0) vm.expectRevert(CannotSubmitZero.selector);
826, 826:          frxETHMinterContract.submitAndDeposit{ value: transfer_amount }(owner, owner);
827, 827:
828, 828:          assertEq(frxETHtoken.balanceOf(owner), 0); /

```



[G-04] Using bools for storage incurs overhead (3 instances)

- Deployment. Gas Saved: **20 221**

- Minumal Method Call. Gas Saved: **266**
- Average Method Call. Gas Saved: **-990**
- Maximum Method Call. Gas Saved: **-5 979**

```
// Booleans are more expensive than uint256 or any type that takes a
// word because each write operation emits an extra SLOAD to find the
// slot's contents, replace the bits taken up by the boolean, and write
// back. This is the compiler's defense against contract upgrades and
// pointer aliasing, and it cannot be disabled.
```

Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas) for the extra SLOAD, and to avoid Gsset (20000 gas) when changing from 'false' to 'true', after having been 'true' in the past



src/ERC20/ERC20PermitPermissionedMint.sol:20

```
diff --git a/src/ERC20/ERC20PermitPermissionedMint.sol b/src/ERC20/ERC20PermitPermissionedMint.sol
index 3bed26d..a5d0aab 100644
--- a/src/ERC20/ERC20PermitPermissionedMint.sol
+++ b/src/ERC20/ERC20PermitPermissionedMint.sol
@@ -17,7 +17,7 @@ contract ERC20PermitPermissionedMint is ERC20Permit {
     17, 17:
     18, 18:         // Minters
     19, 19:         address[] public minters_array; // Allowed to mint
-    20     :-         mapping(address => bool) public minters; // Mapping of address to bool
+    20     20:+         mapping(address => uint256) public minters; // Mapping of address to uint256
     21, 21:
     22, 22:         /* ===== CONSTRUCTOR ===== */
     23, 23:
@@ -43,7 +43,7 @@ contract ERC20PermitPermissionedMint is ERC20Permit {
     43, 43:         }
     44, 44:
     45, 45:         modifier onlyMinters() {
-    46     :-             require(minters[msg.sender] == true, "Only minter can mint");
+    46     46:+             require(minters[msg.sender] == 1, "Only minter can mint");
     47, 47:             _;
     48, 48:         }
     49, 49:
@@ -65,8 +65,8 @@ contract ERC20PermitPermissionedMint is ERC20Permit {
     65, 65:         function addMinter(address minter address) public {
         66, 66:             require(minters[msg.sender] == true, "Only minter can mint");
         67, 67:             minters_array.push(minter);
         68, 68:             minters[minter] = true;
         69, 69:         }
     70, 70:     }
@@ -70,7 +70,7 @@ contract ERC20PermitPermissionedMint is ERC20Permit {
     70, 70:     }
     71, 71:
     72, 72:     function mint(address to, uint256 amount) public {
-    73     :-         require(minters[msg.sender] == true, "Only minter can mint");
+    73     73:+         require(minters[msg.sender] == 1, "Only minter can mint");
     74, 74:         require(to != address(0), "Invalid address");
     75, 75:         require(amount > 0, "Invalid amount");
     76, 76:         require(minters_array.length > 0, "No minter added");
     77, 77:         require(minters_array[0] == msg.sender, "Only minter can mint");
     78, 78:         _mint(to, amount);
     79, 79:     }
@@ -80,7 +80,7 @@ contract ERC20PermitPermissionedMint is ERC20Permit {
     80, 80:     }
     81, 81:
     82, 82:     function burn(uint256 amount) public {
-    83     :-         require(minters[msg.sender] == true, "Only minter can mint");
+    83     83:+         require(minters[msg.sender] == 1, "Only minter can mint");
     84, 84:         require(amount > 0, "Invalid amount");
     85, 85:         _burn(msg.sender, amount);
     86, 86:     }
@@ -87,7 +87,7 @@ contract ERC20PermitPermissionedMint is ERC20Permit {
     87, 87:     }
     88, 88:
     89, 89:     function burnFrom(address from, uint256 amount) public {
-    90     :-         require(minters[msg.sender] == true, "Only minter can mint");
+    90     90:+         require(minters[msg.sender] == 1, "Only minter can mint");
     91, 91:         require(from != address(0), "Invalid address");
     92, 92:         require(amount > 0, "Invalid amount");
     93, 93:         require(minters_array.length > 0, "No minter added");
     94, 94:         require(minters_array[0] == msg.sender, "Only minter can mint");
     95, 95:         _burnFrom(from, amount);
     96, 96:     }
@@ -97,7 +97,7 @@ contract ERC20PermitPermissionedMint is ERC20Permit {
     97, 97:     }
     98, 98:
     99, 99:     function burnFromMinter(address from, uint256 amount) public {
-    100    :-         require(minters[from] == true, "Invalid minter");
+    100    100:+         require(minters[from] == 1, "Invalid minter");
     101, 101:         require(amount > 0, "Invalid amount");
     102, 102:         require(minters_array.length > 0, "No minter added");
     103, 103:         require(minters_array[0] == msg.sender, "Only minter can mint");
     104, 104:         _burnFrom(from, amount);
     105, 105:     }
@@ -107,7 +107,7 @@ contract ERC20PermitPermissionedMint is ERC20Permit {
     107, 107:     }
     108, 108:
     109, 109:     function burnFromMinterAndAllow(address from, uint256 amount, address to) public {
-    110    :-         require(minters[from] == true, "Invalid minter");
+    110    110:+         require(minters[from] == 1, "Invalid minter");
     111, 111:         require(amount > 0, "Invalid amount");
     112, 112:         require(to != address(0), "Invalid address");
     113, 113:         require(minters_array.length > 0, "No minter added");
     114, 114:         require(minters_array[0] == msg.sender, "Only minter can mint");
     115, 115:         _burnFrom(from, amount);
     116, 116:         _mint(to, amount);
     117, 117:     }
@@ -118,7 +118,7 @@ contract ERC20PermitPermissionedMint is ERC20Permit {
     118, 118:     }
     119, 119:
     120, 120:     function burnFromMinterAndAllowMinter(address from, uint256 amount, address to) public {
-    121    :-         require(minters[from] == true, "Invalid minter");
+    121    121:+         require(minters[from] == 1, "Invalid minter");
     122, 122:         require(amount > 0, "Invalid amount");
     123, 123:         require(to != address(0), "Invalid address");
     124, 124:         require(minters_array.length > 0, "No minter added");
     125, 125:         require(minters_array[0] == msg.sender, "Only minter can mint");
     126, 126:         _burnFrom(from, amount);
     127, 127:         _mint(to, amount);
     128, 128:     }
@@ -129,7 +129,7 @@ contract ERC20PermitPermissionedMint is ERC20Permit {
     129, 129:     }
     130, 130:
     131, 131:     function burnFromMinterAndAllowMinterAndAllow(address from, uint256 amount, address to) public {
-    132    :-         require(minters[from] == true, "Invalid minter");
+    132    132:+         require(minters[from] == 1, "Invalid minter");
     133, 133:         require(amount > 0, "Invalid amount");
     134, 134:         require(to != address(0), "Invalid address");
     135, 135:         require(minters_array.length > 0, "No minter added");
     136, 136:         require(minters_array[0] == msg.sender, "Only minter can mint");
     137, 137:         _burnFrom(from, amount);
     138, 138:         _mint(to, amount);
     139, 139:     }
@@ -140,7 +140,7 @@ contract ERC20PermitPermissionedMint is ERC20Permit {
     140, 140:     }
     141, 141:
     142, 142:     function burnFromMinterAndAllowMinterAndAllowMinter(address from, uint256 amount, address to) public {
-    143    :-         require(minters[from] == true, "Invalid minter");
+    143    143:+         require(minters[from] == 1, "Invalid minter");
     144, 144:         require(amount > 0, "Invalid amount");
     145, 145:         require(to != address(0), "Invalid address");
     146, 146:         require(minters_array.length > 0, "No minter added");
     147, 147:         require(minters_array[0] == msg.sender, "Only minter can mint");
     148, 148:         _burnFrom(from, amount);
     149, 149:         _mint(to, amount);
     150, 150:     }
@@ -151,7 +151,7 @@ contract ERC20PermitPermissionedMint is ERC20Permit {
     151, 151:     }
     152, 152:
     153, 153:     function burnFromMinterAndAllowMinterAndAllowMinterAndAllow(address from, uint256 amount, address to) public {
-    154    :-         require(minters[from] == true, "Invalid minter");
+    154    154:+         require(minters[from] == 1, "Invalid minter");
     155, 155:         require(amount > 0, "Invalid amount");
     156, 156:         require(to != address(0), "Invalid address");
     157, 157:         require(minters_array.length > 0, "No minter added");
     158, 158:         require(minters_array[0] == msg.sender, "Only minter can mint");
     159, 159:         _burnFrom(from, amount);
     160, 160:         _mint(to, amount);
     161, 161:     }
@@ -162,7 +162,7 @@ contract ERC20PermitPermissionedMint is ERC20Permit {
     162, 162:     }
     163, 163:
     164, 164:     function burnFromMinterAndAllowMinterAndAllowMinterAndAllowMinter(address from, uint256 amount, address to) public {
-    165    :-         require(minters[from] == true, "Invalid minter");
+    165    165:+         require(minters[from] == 1, "Invalid minter");
     166, 166:         require(amount > 0, "Invalid amount");
     167, 167:         require(to != address(0), "Invalid address");
     168, 168:         require(minters_array.length > 0, "No minter added");
     169, 169:         require(minters_array[0] == msg.sender, "Only minter can mint");
     170, 170:         _burnFrom(from, amount);
     171, 171:         _mint(to, amount);
     172, 172:     }
@@ -173,7 +173,7 @@ contract ERC20PermitPermissionedMint is ERC20Permit {
     173, 173:     }
     174, 174:
     175, 175:     function burnFromMinterAndAllowMinterAndAllowMinterAndAllowMinterAndAllow(address from, uint256 amount, address to) public {
-    176    :-         require(minters[from] == true, "Invalid minter");
+    176    176:+         require(minters[from] == 1, "Invalid minter");
     177, 177:         require(amount > 0, "Invalid amount");
     178, 178:         require(to != address(0), "Invalid address");
     179, 179:         require(minters_array.length > 0, "No minter added");
     180, 180:         require(minters_array[0] == msg.sender, "Only minter can mint");
     181, 181:         _burnFrom(from, amount);
     182, 182:         _mint(to, amount);
     183, 183:     }
@@ -184,7 +184,7 @@ contract ERC20PermitPermissionedMint is ERC20Permit {
     184, 184:     }
     185, 185:
     186, 186:     function burnFromMinterAndAllowMinterAndAllowMinterAndAllowMinterAndAllowMinter(address from, uint256 amount, address to) public {
-    187    :-         require(minters[from] == true, "Invalid minter");
+    187    187:+         require(minters[from] == 1, "Invalid minter");
     188, 188:         require(amount > 0, "Invalid amount");
     189, 189:         require(to != address(0), "Invalid address");
     190, 190:         require(minters_array.length > 0, "No minter added");
     191, 191:         require(minters_array[0] == msg.sender, "Only minter can mint");
     192, 192:         _burnFrom(from, amount);
     193, 193:         _mint(to, amount);
     194, 194:     }
@@ -195,7 +195,7 @@ contract ERC20PermitPermissionedMint is ERC20Permit {
     195, 195:     }
     196, 196:
     197, 19
```

```

66, 66:         require(minter_address != address(0), "Zero
67, 67:
- 68     :-         require(minters[minter_address] == false, "Z
- 69     :-         minters[minter_address] = true;
+       68:+         require(minters[minter_address] == 0, "Addre
+       69:+         minters[minter_address] = 1;
70, 70:         minters_array.push(minter_address);
71, 71:
72, 72:         emit MinterAdded(minter_address);
@@ -75,7 +75,7 @@ contract ERC20PermitPermissionedMint is ERC20F
75, 75:         // Remove a minter
76, 76:         function removeMinter(address minter_address) pu
77, 77:         require(minter_address != address(0), "Zero
- 78     :-         require(minters[minter_address] == true, "Ac
+       78:+         require(minters[minter_address] == 1, "Addre
79, 79:
80, 80:         // Delete from the mapping
81, 81:         delete minters[minter_address];

```



src/frxETHMinter.sol:43, 49-50

```

diff --git a/src/frxETHMinter.sol b/src/frxETHMinter.sol
index 4565883..3036cea 100644
--- a/src/frxETHMinter.sol
+++ b/src/frxETHMinter.sol
@@ -40,14 +40,14 @@ contract frxETHMinter is OperatorRegistry, F
40, 40:
41, 41:         uint256 public withholdRatio; // What we keep ar
42, 42:         uint256 public currentWithheldETH; // Needed for
- 43     :-         mapping(bytes => bool) public activeValidators;
+       43:+         mapping(bytes => uint256) public activeValidator
44, 44:
45, 45:         IDepositContract public immutable depositContract
46, 46:         frxETH public immutable frxETHToken;
47, 47:         IsfrxETH public immutable sfrxETHToken;
48, 48:
- 49     :-         bool public submitPaused;
- 50     :-         bool public depositEtherPaused;
+       49:+         uint256 public submitPaused;
+       50:+         uint256 public depositEtherPaused;
51, 51:
52, 52:         constructor(
53, 53:             address depositContractAddress,

```

```

@@ -84,7 +84,7 @@ contract frxETHMinter is OperatorRegistry, Ree
    84, 84:      /// @notice Mint frxETH to the recipient using s
    85, 85:      function _submit(address recipient) internal nor
    86, 86:      // Initial pause and value checks
-   87      :-      require(!submitPaused, "Submit is paused");
+   87      +:      require(0==submitPaused, "Submit is paused")
    88, 88:      require(msg.value != 0, "Cannot submit 0");
    89, 89:
    90, 90:      // Give the sender frxETH
@@ -119,7 +119,7 @@ contract frxETHMinter is OperatorRegistry, F
    119, 119:     /// @dev Usually a bot will call this periodical
    120, 120:     function depositEther() external nonReentrant {
    121, 121:     // Initial pause check
-   122      :-      require(!depositEtherPaused, "Depositing ETH
+   122      +:      require(0==depositEtherPaused, "Depositing E
    123, 123:
    124, 124:     // See how many deposits can be made. Trunca
    125, 125:     uint256 numDeposits = (address(this).balance
@@ -137,7 +137,7 @@ contract frxETHMinter is OperatorRegistry, F
    137, 137:
    138, 138:     // Make sure the validator hasn't been c
    139, 139:     // until withdrawals are allowed
-   140      :-      require(!activeValidators[pubKey], "Vali
+   140      +:      require(0==activeValidators[pubKey], "Va
    141, 141:
    142, 142:     // Deposit the ether in the ETH 2.0 depe
    143, 143:     depositContract.deposit{value: DEPOSIT_S
@@ -148,7 +148,7 @@ contract frxETHMinter is OperatorRegistry, F
    148, 148:     );
    149, 149:
    150, 150:     // Set the validator as used so it won't
-   151      :-      activeValidators[pubKey] = true;
+   151      +:      activeValidators[pubKey] = 1;
    152, 152:
    153, 153:     emit DepositSent(pubKey, withdrawalCrede
    154, 154:     }
@@ -175,14 +175,14 @@ contract frxETHMinter is OperatorRegistry,
    175, 175:
    176, 176:     /// @notice Toggle allowing submites
    177, 177:     function togglePauseSubmits() external onlyByOwr
-   178      :-      submitPaused = !submitPaused;
+   178      +:      submitPaused = submitPaused==1?0:1;
    179, 179:
    180, 180:     emit SubmitPaused(submitPaused);
    181, 181:     }
    182, 182:

```

```

183, 183:      /// @notice Toggle allowing depositing ETH to va
184, 184:      function togglePauseDepositEther() external only
- 185      :-      depositEtherPaused = !depositEtherPaused;
+      185:+      depositEtherPaused = depositEtherPaused==1?(
186, 186:
187, 187:          emit DepositEtherPaused(depositEtherPaused);
188, 188:      }
@@ -205,9 +205,9 @@ contract frxETHMinter is OperatorRegistry, F
205, 205:      event EmergencyEtherRecovered(uint256 amount);
206, 206:      event EmergencyERC20Recovered(address tokenAddre
207, 207:      event ETHSubmitted(address indexed sender, addre
- 208      :-      event DepositEtherPaused(bool new_status);
+      208:+      event DepositEtherPaused(uint256 new_status);
209, 209:      event DepositSent(bytes indexed pubKey, bytes wi
- 210      :-      event SubmitPaused(bool new_status);
+      210:+      event SubmitPaused(uint256 new_status);
211, 211:      event WithheldETHMoved(address indexed to, uint2
212, 212:      event WithholdRatioSet(uint256 newRatio);
213, 213: }

```



[G-05] Unchecking arithmetics operations that can't underflow/overflow (7 instances)

- Deployment. Gas Saved: **18 621**
- Minumal Method Call. Gas Saved: **227**
- Average Method Call. Gas Saved: **503**
- Maximum Method Call. Gas Saved: **829**

Solidity version 0.8+ comes with implicit overflow and underflow checks on unsigned integers. When an overflow or an underflow isn't possible (as an example, when a comparison is made before the arithmetic operation), some gas can be saved by using an unchecked block:

<https://docs.soliditylang.org/en/v0.8.10/control-structures.html#checked-or-unchecked-arithmetic>



src/ERC20/ERC20PermitPermissionedMint.sol:84

```
diff --git a/src/ERC20/ERC20PermitPermissionedMint.sol b/src/ERC
```

```

index 3bed26d..25010cb 100644
--- a/src/ERC20/ERC20PermitPermissionedMint.sol
+++ b/src/ERC20/ERC20PermitPermissionedMint.sol
@@ -81,11 +81,14 @@ contract ERC20PermitPermissionedMint is ERC2
    81, 81:          delete minters[minter_address];
    82, 82:
    83, 83:          // 'Delete' from the array by setting the ac
-   84      :-          for (uint i = 0; i < minters_array.length; i
+   84      84:+          for (uint i = 0; i < minters_array.length;){
    85, 85:              if (minters_array[i] == minter_address)
    86, 86:                  minters_array[i] = address(0); // Th
    87, 87:                  break;
    88, 88:              }
+   89      89:+          unchecked {
+   90      90:+              ++i;
+   91      91:+          }
    89, 92:          }
    90, 93:
    91, 94:          emit MinterRemoved(minter_address);

```



src/OperatorRegistry.sol:63, 84, 114

```

diff --git a/src/OperatorRegistry.sol b/src/OperatorRegistry.sol
index f81094c..aef4e17 100644
--- a/src/OperatorRegistry.sol
+++ b/src/OperatorRegistry.sol
@@ -60,8 +60,11 @@ contract OperatorRegistry is Owned {
    60, 60:          Reason we don't do that here is for gas */
    61, 61:          function addValidators(Validator[] calldata vali
    62, 62:              uint arrayLength = validatorArray.length;
-   63      :-          for (uint256 i = 0; i < arrayLength; ++i) {
+   63      63:+          for (uint256 i = 0; i < arrayLength;){
    64, 64:              addValidator(validatorArray[i]);
+   65      65:+              unchecked {
+   66      66:+                  ++i;
+   67      67:+              }
    65, 68:          }
    66, 69:      }
    67, 70:
@@ -81,8 +84,11 @@ contract OperatorRegistry is Owned {
    81, 84:          /// @notice Remove validators from the end of th
    82, 85:          function popValidators(uint256 times) public onl
    83, 86:              // Loop through and remove validator entries

```

```

- 84      :-      for (uint256 i = 0; i < times; ++i) {
+          87:++      for (uint256 i = 0; i < times;) {
      85, 88:          validators.pop();
+          89:++          unchecked {
+          90:++              ++i;
+          91:++          }
      86, 92:      }
      87, 93:
      88, 94:          emit ValidatorsPopped(times);
@@ -111,10 +117,13 @@ contract OperatorRegistry is Owned {
      111, 117:          delete validators;
      112, 118:
      113, 119:          // Fill the new validators array with al
- 114      :-      for (uint256 i = 0; i < original_validat
+          120:++      for (uint256 i = 0; i < original_validat
      115, 121:          if (i != remove_idx) {
      116, 122:              validators.push(original_validat
      117, 123:          }
+          124:++          unchecked {
+          125:++              ++i;
+          126:++          }
      118, 127:      }
      119, 128:      }
      120, 129:

```



src/frxETHMinter.sol:96, 129

```

diff --git a/src/frxETHMinter.sol b/src/frxETHMinter.sol
index 4565883..4cee757 100644
--- a/src/frxETHMinter.sol
+++ b/src/frxETHMinter.sol
@@ -93,7 +93,9 @@ contract frxETHMinter is OperatorRegistry, Rec
      93, 93:          // Track the amount of ETH that we are keepi
      94, 94:          uint256 withheld_amt = 0;
      95, 95:          if (withholdRatio != 0) {
- 96      :-      withheld_amt = (msg.value * withholdRati
+          96:++          unchecked {
+          97:++              withheld_amt = (msg.value * withholc
+          98:++          }
      97, 99:          currentWithheldETH += withheld_amt;
      98, 100:      }
      99, 101:
@@ -126,7 +128,7 @@ contract frxETHMinter is OperatorRegistry, F

```

```

126, 128:                require(numDeposits > 0, "Not enough ETH in
127, 129:
128, 130:                // Give each deposit chunk to an empty valid
- 129      :-            for (uint256 i = 0; i < numDeposits; ++i) {
+      131:+            for (uint256 i = 0; i < numDeposits;) {
130, 132:                // Get validator information
131, 133:                (
132, 134:                bytes memory pubKey,
@@ -151,6 +153,9 @@ contract frxETHMinter is OperatorRegistry, F
151, 153:                activeValidators[pubKey] = true;
152, 154:
153, 155:                emit DepositSent(pubKey, withdrawalCred
+      156:+            unchecked {
+      157:+                ++i;
+      158:+            }
154, 159:            }
155, 160:        }
156, 161:

```



[G-06] storage **pointer to a structure is cheaper than copying each value of the structure into memory** , same for array **and** mapping **(1 instance)**

- Deployment. Gas Saved: **8 208**
- Minumal Method Call. Gas Saved: **106**
- Average Method Call. Gas Saved: **-970**
- Maximum Method Call. Gas Saved: **2 487**



src/OperatorRegistry.sol:161

```

diff --git a/src/OperatorRegistry.sol b/src/OperatorRegistry.sol
index f81094c..b7b094d 100644
--- a/src/OperatorRegistry.sol
+++ b/src/OperatorRegistry.sol
@@ -158,7 +158,7 @@ contract OperatorRegistry is Owned {
158, 158:                bytes32 depositDataRoot
159, 159:            )
160, 160:        {
- 161      :-            Validator memory v = validators[i];

```

```

+      161: +      Validator storage v = validators[i];
      162, 162:
      163, 163:      // Return the validator's information
      164, 164:      pubKey = v.pubKey;

```



[G-07] $x = x + y$ is more efficient, than $x += y$ (4 instances)

- Deployment. Gas Saved: **5 007**
- Minumal Method Call. Gas Saved: **82**
- Average Method Call. Gas Saved: **87**
- Maximum Method Call. Gas Saved: **101**



src/frxETHMinter.sol:97, 168

```

diff --git a/src/frxETHMinter.sol b/src/frxETHMinter.sol
index 4565883..a591be9 100644
--- a/src/frxETHMinter.sol
+++ b/src/frxETHMinter.sol
@@ -94,7 +94,7 @@ contract frxETHMinter is OperatorRegistry, Rec
     94,  94:         uint256 withheld_amt = 0;
     95,  95:         if (withholdRatio != 0) {
     96,  96:             withheld_amt = (msg.value * withholdRati
-  97      :-             currentWithheldETH += withheld_amt;
+      97: +             currentWithheldETH = currentWithheldETH
     98,  98:         }
     99,  99:
    100, 100:         emit ETHSubmitted(msg.sender, recipient, msg
@@ -165,7 +165,7 @@ contract frxETHMinter is OperatorRegistry, F
    165, 165:         /// @notice Give the withheld ETH to the "to" ac
    166, 166:         function moveWithheldETH(address payable to, uir
    167, 167:             require(amount <= currentWithheldETH, "Not e
-  168      :-             currentWithheldETH -= amount;
+      168: +             currentWithheldETH = currentWithheldETH - an
    169, 169:
    170, 170:             (bool success,) = payable(to).call{ value: a
    171, 171:             require(success, "Invalid transfer");

```




```

diff --git a/src/xERC4626.sol b/src/xERC4626.sol
index a8a4726..dea5982 100644
--- a/src/xERC4626.sol
+++ b/src/xERC4626.sol
@@ -64,12 +64,12 @@ abstract contract xERC4626 is IxERC4626, ERC
    64,    64:        // Update storedTotalAssets on withdraw/redeem
    65,    65:        function beforeWithdraw(uint256 amount, uint256
    66,    66:            super.beforeWithdraw(amount, shares);
-   67        :-            storedTotalAssets -= amount;
+   67    67: +            storedTotalAssets = storedTotalAssets - amou
    68,    68:        }
    69,    69:
    70,    70:        // Update storedTotalAssets on deposit/mint
    71,    71:        function afterDeposit(uint256 amount, uint256 sh
-   72        :-            storedTotalAssets += amount;
+   72    72: +            storedTotalAssets = storedTotalAssets + amou
    73,    73:            super.afterDeposit(amount, shares);
    74,    74:        }
    75,    75:

```



[G-08] It costs more gas to initialize non-constant/non-immutable variables to zero than to let the default of zero be applied (2 instances)

- Deployment. Gas Saved: **4 415**
- Minumal Method Call. Gas Saved: **0**
- Average Method Call. Gas Saved: **0**
- Maximum Method Call. Gas Saved: **0**

If a variable is not set/initialized, it is assumed to have the default value (0 for uint, false for bool, address(0) for address...). Explicitly initializing it with its default value is an anti-pattern and wastes gas.



src/frxETHMinter.sol:63-64

```

diff --git a/src/frxETHMinter.sol b/src/frxETHMinter.sol

```

```

index 4565883..b0f66a8 100644
--- a/src/frxETHMinter.sol
+++ b/src/frxETHMinter.sol
@@ -60,8 +60,6 @@ contract frxETHMinter is OperatorRegistry, Rec
    60,    60:            depositContract = IDepositContract(depositCo
    61,    61:            frxETHToken = frxETH(frxETHAddress);
    62,    62:            sfrxETHToken = IsfrxETH(sfrxETHAddress);
-   63      :-            withholdRatio = 0; // No ETH is withheld ini
-   64      :-            currentWithheldETH = 0;
    65,    63:            }
    66,    64:
    67,    65:            /// @notice Mint frxETH and deposit it to receiv

```



[G-09] Don't compare boolean expressions to boolean literals (3 instances)

- Deployment. Gas Saved: **3 006**
- Minumal Method Call. Gas Saved: **43**
- Average Method Call. Gas Saved: **-477**
- Maximum Method Call. Gas Saved: **55**



src/ERC20/ERC20PermitPermissionedMint.sol:46, 68, 78

```

diff --git a/src/ERC20/ERC20PermitPermissionedMint.sol b/src/ERC
index 3bed26d..860d2c4 100644
--- a/src/ERC20/ERC20PermitPermissionedMint.sol
+++ b/src/ERC20/ERC20PermitPermissionedMint.sol
@@ -43,7 +43,7 @@ contract ERC20PermitPermissionedMint is ERC20F
    43,    43:            }
    44,    44:
    45,    45:            modifier onlyMinters() {
-   46      :-            require(minters[msg.sender] == true, "Only mi
+   46      46:+            require(minters[msg.sender], "Only minters");
    47,    47:            _;
    48,    48:            }
    49,    49:
@@ -65,7 +65,7 @@ contract ERC20PermitPermissionedMint is ERC20F
    65,    65:            function addMinter(address minter_address) publi
    66,    66:            require(minter_address != address(0), "Zero
    67,    67:

```

```

- 68      :-      require(minters[minter_address] == false, "Z
+      68:+      require(!minters[minter_address], "Address a
69, 69:      minters[minter_address] = true;
70, 70:      minters_array.push(minter_address);
71, 71:
@@ -75,7 +75,7 @@ contract ERC20PermitPermissionedMint is ERC20I
75, 75:      // Remove a minter
76, 76:      function removeMinter(address minter_address) pu
77, 77:      require(minter_address != address(0), "Zero
- 78      :-      require(minters[minter_address] == true, "Ac
+      78:+      require(minters[minter_address], "Address nc
79, 79:
80, 80:      // Delete from the mapping
81, 81:      delete minters[minter_address];

```



[G-10] State variables should be cached in stack variables rather than re-reading them from storage (1 instances)

- Deployment. Gas Saved: **400**
- Minumal Method Call. Gas Saved: **-21**
- Average Method Call. Gas Saved: **511**
- Maximum Method Call. Gas Saved: **4 839**



src/frxETHMinter.sol:95-96

```

diff --git a/src/frxETHMinter.sol b/src/frxETHMinter.sol
index 4565883..802e94b 100644
--- a/src/frxETHMinter.sol
+++ b/src/frxETHMinter.sol
@@ -92,8 +92,9 @@ contract frxETHMinter is OperatorRegistry, Rec
92, 92:
93, 93:      // Track the amount of ETH that we are keepi
94, 94:      uint256 withheld_amt = 0;
- 95      :-      if (withholdRatio != 0) {
- 96      :-          withheld_amt = (msg.value * withholdRati
+      95:+      uint256 _withholdRatio;
+      96:+      if ((_withholdRatio = withholdRatio) != 0) {
+      97:+          withheld_amt = (msg.value * _withholdRat
97, 98:          currentWithheldETH += withheld_amt;
98, 99:      }

```


- Deployment. Gas Saved: **419 688**
- Minumal Method Call. Gas Saved: **5 474**
- Average Method Call. Gas Saved: **270 705**
- Maximum Method Call. Gas Saved: **539 594**

```
diff --git a/original.txt b/foundry.txt
index 83cd313..4a4aaa0 100644
--- a/original.txt
+++ b/foundry.txt
@@ -3,13 +3,13 @@
```

	Deployment Cost	Deployment Size		
-	1439975	7889		
+	1353480	7457		
	Function Name	min	avg	
	DOMAIN_SEPARATOR	365	365	
-	addMinter	46593	59107	
+	addMinter	46508	59022	
	allowance	826	1048	
@@ -19,7 +19,7 @@				
	decimals	289	289	
-	minter_mint	4906	37627	
+	minter_mint	4918	37639	
	nonces	661	1751	
@@ -45,45 +45,45 @@				

	Deployment Cost	Deployment Size
-	2575261	13642
+	2242068	11990



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

An open organization | [Twitter](#) | [Discord](#) | [GitHub](#) | [Medium](#) | [Newsletter](#) | [Media kit](#) | [Careers](#) | [code4rena.eth](#)