Kiln 2.2 Security Review

Auditors

Saw-mon and Natalie, Lead Security Researcher

Optimum, Lead Security Researcher

Chris Smith, Lead Security Researcher

Akshay Srivastav, Security Researcher

Report prepared by: Lucas Goiriz

November 12, 2024

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Kiln is a staking platform you can use to stake directly, or whitelabel staking into your product. It enables users to stake crypto assets, manually or programmatically, while maintaining custody of your funds in your existing solution, such Fireblocks, Copper, or Ledger.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of vsuite according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 14 days in total, Kiln engaged with Spearbit to review the vsuite protocol. In this period of time a total of **57** issues were found.

Summary

Project Name	Kiln
Repository	vsuite
Commit	194d71...73c3
Type of Project	Enterprise Staking, DeFi
Audit Timeline	Apr 22 to May 3
Fix period	May 4 - May 6

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	3	3	0
Medium Risk	3	3	0
Low Risk	15	14	1
Gas Optimizations	7	6	1
Informational	29	25	4
Total	**57**	**51**	**6**

5 Findings

5.1 High Risk

5.1.1 Incorrect rebates distribution

Severity: High Risk

Context: vNFTAct.driver.sol#L454

Description: The `vNFTActDriver._actionExit` incorrectly sets the rebate value to the wrong key (`tokenId` instead of `validatorId`) during the user's exit:

```
rebates[tokenId] = REBATE_ON_EXIT;
```

Scenario:

- Alice minted token id 1 which points to validator id 1.
- Bob morphs Alice's token id by calling `onRewards`. Alice's token id is now 2.
- Alice performs exit. The rebate gets set to token id 2 (i.e. validator id 2).
- Bob mints the NFT of token id 3, his validator id is 2.
- Now when Alice's funds arrive from CL she won't get any rebate and will pay commission on entire ETH amount.
- As bob receives CL/EL rewards, his rewards won't be charged any commission due to the incorrect rebate set by Alice's exit.
- After Bob exhaust the 32 ETH rebate, he can perform his own exit and his rebate can again be set to 32 ETH.

In this scenario Alice did not get any rebate and Bob got double rebate. The scenario can be performed by other users to gain extra rebates.

Recommendation: Consider changing the implementation to this:

```
- rebates[tokenId] = REBATE_ON_EXIT;
+ rebates[validatorIds[idx]] = REBATE_ON_EXIT;
```

Kiln: Resolved by PR 157.

Spearbit: Rebates are now correctly assigned to `validatorId` which fixes the issue.

5.1.2 `vNFT.internals._morph` does not set approvals for new token leading to front run DoS for spenders

Severity: High Risk

Context: vNFT.internals.sol#L157

Description: When a `tokenId` is morphed during `onRewards` in a static system, it does not transfer the exiting `$tokenApprovals` to the `newTokenId`.

Because the `onRewards` is callable by anyone, an attacker frontrun a transaction from a spender to transfer a token by sending 1 wei of ETH to `onRewards` causing the tokenId to morph and the spenders approval to be lost.

The impact of this is any actions that require NFT approval (such as selling, staking, etc) are blockable by a malicious actor.

Recommendation: Add

```
$tokenApprovals.get()[newTokenId] = $tokenApprovals.get()[tokenId];
```

to the `_morph` function.

Kiln: As the system should clear approvals on morphing, the fix here was to make `onRewards` permissioned, preventing the DoS vector. Resolved by PR 158.

Spearbit: This fix works and maintains the desired state of the system (i.e. clearing approvals on morphing while not allowing griefing).

5.1.3 `vNFT.onRewards`: Attackers can "morph" any token held by others potentially censoring transfers in case `config.mode == vNFTMode.Static`

Severity: High Risk

Context: vNFT.sol#L205

Description: In the current version of the code, `vNFT.onRewards` is callable by anyone with `msg.value > 0`. Attackers might abuse it to call `onRewards` with a `validatorId` that belongs to a victim and by doing this cause the victim's `tokenId` to change without his consent nor awareness. This kind of attack may cause confusion for users since their token ids will be changed unexpectedly which (depends on the wallet support) might look like they have lost the token. Additionally, this kind of attack can be used in a front-running manner to actively censor any attempt to transfer the token by the user. Another potential attack (although much less severe) will be to call `onRewards` with a non existing `validatorId`. It will cause both `$burnCounter` and `$mintCounter` to be artificially increased by 1. It is important to mention that `totalSupply` would not be impacted since it is always equal to `$mintCounter - $burnCounter`.

Recommendation: To properly mitigate this issue we will need to make sure `vNFT.onRewards` is accessible only by the `MinimalRecipient` that is supposed to call it. The elegant way to do that will be for the `vNFT.onRewards` to only accept calls from the `vFactory` contract, then `vFactory._withdraw` should store the `withdrawalRecipient` in a storage variable (transient storage variable will be more gas efficient) before the call to `withdrawalRecipient.exec` and set it back to `address(0)` after. In addition, the first argument of the call to `withdrawalRecipient.exec` should be the `vFactory` address instead and the `vFactory` should implement a callback function that is only callable by the previously stored `withdrawalRecipient` and that will finally call `vNFT.onRewards`.

Kiln: Fixed in commit 35ec1dcb.

Spearbit: Fixed by implementing the overall solution proposed by the auditors, with a few slight changes that achieve the same result. It is important to add that in the given mitigation commit hash a new feature was introduced to let users that request a withdraw to specify another recipient that is stored in `nextRecipient` with an option for the withdrawal flow to skip the `vNFT` contract `onRewards` function.

5.2 Medium Risk

5.2.1 In some rare cases due to collision in truncation of token owners, approval of `vNFT` can be stolen

Severity: Medium Risk

Context: vNFT.sol#L266-L277, NFT.sol#L73-L84, vNFT.sol#L435-L437, vNFT.internals.sol#L317

Description: The overwritten `vNFT.approve` function is almost identical to its inherited `NFT.approve` except:

1. It has a `nonReentrant` modifier
2. Its `_ownerOf` function is also overwritten in `vNFT` although the both implementations give the same result or truncation `bytes32` to a `address`:

```
function approve(address to, uint256 tokenId) external ... {
    address owner = _ownerOf(tokenId);
    if (to == owner) {
        revert ApprovalToOwner(owner);
    }

    if (msg.sender != owner && !_isApprovedForAll(owner, msg.sender)) {
        revert LibErrors.Unauthorized(msg.sender, owner);
    }

    _approve(to, owner, tokenId);
}
```

The important fact is that `_ownerOf(tokenId)` truncates the actual owner:

```
// address(uint160(uint256(bytes32($owners.get()[tokenId]))))
$owners.get()[tokenId].toBytes32().getAddress()
```

Now this causes the following issue, if the `tokenId` was owned by a group the above truncation operation gets ride of the group bit flag in `$owners.get()[tokenId]` and thus potentially can collide with another address or owner of a different token.

Assume that:

1. `$owners.get()[tokenId] = 0b100...0 <ADDRESS>` is group *G* owning the token id *I* and the above truncation in `_ownerOf` for this pair (*G*, *i*) gives address *A* and the root owner of of the group *G* is address *B*.
2. Then *A* can set approval for this token *I* to any desired address when `approve` is called and the desired address can transfer this token id from the group *G* to any other desired address.

This might be a rare occurrence since:

- Group raw ids start from `0x1100000000000000000000000000000000000000 + 1`.
- The attacker might have to farm for the truncated group raw id / address.
- The used addresses by the attacker might need enough authorised rights to perform that approval and transfer.

Note that the attacker can also frontrun the creation of the original group *G* by a sequence of group creations so that the truncation address falls on a specified farmed address (might require a lot of gas).

Proof of Concept: Add the following test case to `vNFTInitTest` and run it with `forge t -vvvv --mt test_ADDRESS_GROUP_SPACE_APPROVAL_COLLISSION`:

```
function _genGivenAddr(string memory name, address addr) internal returns (address) {
    vm.label(addr, name);
    return addr;
}

function test_ADDRESS_GROUP_SPACE_APPROVAL_COLLISSION(uint256 _salt) external {
    vnft = _deploy_vNFT(0, _salt % 2 == 0 ? STATIC : DYNAMIC);
    TestValidationKeyRegistry memory tvkr = sudo.factoryAdmin__setupKeys(o, 128, bytes32(0));
    IvNFTAct.PurchaseParameters memory pp = _getPurchaseParameters(tvkr, 1);

    bytes32 groupId = vnft.nextGroupId(0);
    uint256 tokenId = 1;

    {
        // User A purchases a validator
        address purchaser = _genGivenAddr("purchaser", address(2));
        vm.deal(purchaser, 32 ether);
```

```
        vm.stopPrank();
        vm.startPrank(purchaser);
        vnft.purchase{value: 32 ether}(pp);

        expect(vnft.balanceOf(purchaser)).toEqual(1);

        // create a group that would truncate to User B's address
        IvNFTAct.ActionGroup[] memory ag = new IvNFTAct.ActionGroup[](1);
        ag[0] = IvNFTAct.ActionGroup({actions: new IvNFTAct.Action[](1), tokenIds: new uint256[](0),
↪   contextValidatorIds: false});
        ag[0].actions[0] = IvNFTAct.Action({
            actionType: IvNFTAct.ActionType.CreateGroup,
            data: abi.encode(IvNFTAct.CreateGroupParameters({recipient:
↪   Recipient.fromAddress(purchaser)}))
        });

        vnft.act(ag);

        // transfer ownership to the created group
        vnft.act(_actTransferParameters(Recipient.fromAddress(purchaser), groupId, tokenId));
    }

    {
        // User B
        address attacker = _genGivenAddr("attacker", Recipient.getAddress(groupId));
        address attacker2 = _genGivenAddr("attacker", address(3));
        address attacker3 = genAddr("attacker3", _salt);

        vm.stopPrank();
        vm.startPrank(attacker);
        vnft.approve(attacker2, tokenId);

        vm.stopPrank();
        vm.startPrank(attacker2);
        vnft.act(_actTransferParameters(groupId, Recipient.fromAddress(attacker3), tokenId));
    }
}
```

Recommendation: Either:

- Disallow `approve` for token ids owned by group owners or...
- Instead of checking against `_ownerOf(tokenId)` use `_tokenResolvedOwner(tokenId, MAX_OWNERSHIP_DEPTH)`.

Kiln: Fixed in PR 164 using the second recommendation.

Spearbit: Verified.

5.2.2 `vNFTActDriver._actionExit`: Morphing allows Double Exit Validator from vNFT and double counting rebates

Severity: Medium Risk

Context: vNFTAct.driver.sol#L453-L454, vNFT.internals.sol#L165-L166

Description: The `_morph` function is called when `_actionClaim` is used for validators in a `STATIC` system. The consequence of this action is that the token ids that are passed in from `_actGroup` are morphed. The current `validators[tokenId]` is copied to a `validators[newTokenId]`. However, this does not change the id that is in the current array of `tokenIds` inside that `_actGroup`. Since user's are able to pass multiple actions to the same act group, they can double exit their validators.

Proof of Concept:

1. Create action group with two actions: Claim and Exit with your TokenId, the Claim morph's the `tokenId` but the exit sets `rebate` and `.exited` based on the old `tokenId`.
2. Call Claim on the validator during the unlock/exit period to claim some of the rewards which will include the rebate on exiting.
3. Later (once the validator's ETH has arrived) submit a second action group that exits with the morphed TokenId and then Claim. The exit resets the `REBATE` for the new tokenId allowing double rebating some portion of the exiting.

Impact:

1. The `vFactory` will emit two Exit events for that validator Id. *Impact unknown since how these events are consumed and used is beyond our scope*.
2. In the last group, exit the morphed tokenId. This was not marked as exited in the first group, so exit happens and the `rebates` are reset to `REBATE_ON_EXIT` which will give the user a greater than intended rebate on the commission in the Claim $\rightarrow$ `onRewards` flow.

Note in the current code, there is a bug where exit sets `rebates[tokenId] = REBATE_ON_EXIT`. This current code would avoid this underpaying on the commission; however, it is not correct since the key for `rebates` is `validatorId`. Once this bug is corrected, the code will be vulnerable to user's paying less than they should in their commission.

Recommendation: The act groups and actions are a sophisticated, but very complex system. There needs to be additional testing for scenarios with multiple actions per group and different combinations of groups/actions. As one example, the path of `recipient != msgSender` in creating a group does not seem to have test coverage. Ideally, you would find a way to build this into the forge stateful fuzzy (invariants) so you get the benefit of that suite making random calls in random sequences.

Specifically for this scenario: you might need to investigate ways you could signal to an actGroup that its tokenId array has been morphed (possible in `_pushMorphed` or as a return of `_actionClaim`.

Kiln: Resolved by PR 160.

Spearbit: This prevents morphing exited validators which resolves this immediate issue. However, this PR does not add additional testing for this particular issue or other complex act groups/action sequences. We would still recommend improving the test scenarios as that might expose other interaction issues.

5.2.3 `vNFTActDriver`: Sanctioned users can use a delegate to set extra data and threshold values

Severity: Medium Risk

Context: vNFTAct.driver.sol#L537, vNFTAct.driver.sol#L557

Description: Both `_actionSetExtraData` and `_actionSetThreshold` allow a delegate to call these functions on behalf of a beneficiary. During the execution of both these functions only the `msg.sender` is checked against the sanctions list which means that a sanctioned beneficiary can use a non-sanctioned delegate to call these functions and bypass this restriction.

Recommendation: Consider adding another call to `_checkRightsAndSanctions` to make sure that sanctioned beneficiaries are not allowed to call these functions.

Kiln: Fixed in PR 161.

Spearbit: Verified.

5.3 Low Risk

5.3.1 vNFTs whose validators are exited can still be morphed

Severity: Low Risk

Context: vNFT.internals.sol#L157

Description: The `vNFTInternal._morph` function does not validate that the validator of the NFT should not be already exited. Due to this the vNFTs whose validators are already exited can still be morphed.

Recommendation: Only perform morphing when the `vNFTValidator.exited` flag of `tokenId` is `false`.

Kiln: Resolved in PR 160.

Spearbit: As suggested, now morphing is only performed for non-exited validators.

5.3.2 Missing vNFT `tokenId` existence check in `vNFTInternals._morph` function

Severity: Low Risk

Context: vNFT.internals.sol#L158

Description: The `_morph` function is missing to validate that the `tokenId` must exist. Due to which the `_morph` function can be invoked for any `validatorId`. This will lead to arbitrary value getting stored in the `$validatorIdToTokenId` state.

The cost of this type of attack is at least `1` wei plus any gas costs related to calling `vNFT.onRewards`.

Proof of Concept: Add this test case to `vNFTInitTest` and run it with `forge t -vvvv --mt test_onRewards_morph`:

```
event Morphed(uint256 validatorId, uint256 oldTokenId, uint256 newTokenId);

function test_onRewards_morph(uint256 validatorId) public {
    vnft = _deploy_vNFT(0, STATIC);

    vm.expectEmit(true, true, true, true);
    emit Morphed(validatorId, 0, 1);
    vnft.onRewards{value: 1}(validatorId);
}
```

```
[122628] vNFT::onRewards{value: 1}(17785 [1.778e4])
  emit Transfer(from: 0x0000000000000000000000000000000000000000, to:
↪  0x0000000000000000000000000000000000000000, tokenId: 0)
  emit Morphed(validatorId: 17785 [1.778e4], oldTokenId: 0, newTokenId: 1)
  emit Transfer(from: 0x0000000000000000000000000000000000000000, to:
↪  0x0000000000000000000000000000000000000000, tokenId: 1)
  [0] 0x0000000000000000000000000000000000000000::fallback{value: 1}()
    ← ()
  emit RewardsPayout(recipient: 0x0000000000000000000000000000000000000000, amount: 1)
  ← ()
```

Recommendation: Consider adding this check:

```
  function _morph(uint256 validatorId) internal {
      uint256 tokenId = $validatorIdToTokenId.get()[validatorId];
+     _requireExists(tokenId);
      // ...
  }
```

Kiln: Resolved by PR 162.

Spearbit: The check has been added.

5.3.3 Execution of `ActionGroup` batch can be forcefully reverted

Severity: Low Risk

Context: vNFTAct.driver.sol#L580

Description: The `act` function supports `Authorize` action type. Authorization is done by using signature signed by `$authorizer` (or `$admin`). Also a single signature can only be used once due to the use of `$nonces`.

In case an `ActionGroup` batch contains an `Authorize` action type then an attacker can frontrun the batch, extract the `AuthorizeParameters` and submit that in the frontrunning transaction. Due to which the execution of original `ActionGroup` batch will get reverted.

Any batch containing an `Authorize` action can be forcefully reverted using this attack.

Recommendation: This could be an accepted risk. But in case Kiln wants to fix this then a `submitter` address can be attached in the signature which ensures that only the designated submitter can submit the signature. Or, the `Authorize` action can be made a no-op in case it was already frontrun.

Kiln 2.2: Resolved by PR 162. Added a no-op when verifying an applying authorization if rights are not going to change.

Spearbit: Verified.

5.3.4 `MultiPool20` allows unauthorized (`RIGHTS__TRANSFER`) `transferFrom`

Severity: Low Risk

Context: MultiPool20.sol#L551-L556

Description: The documentation PR states that:

> `RIGHTS__TRANSFER` is required when the `msg.sender` is not the recipient of the funds or tokens ... It is necessary for the caller and recipient of `transfer()` or `transferFrom()`.

However, it is not checked in the `_checkTransferRights` function which only checks that the `to` and `from` address have transfer rights.

Recommendation: Add (along with an appropriate `msg.sender != to/from` check):

```
if (senderRights & MultiPool.RIGHTS__TRANSFER == 0) {
    revert MissingAuthorizations(fromRights.toUint248(), MultiPool.RIGHTS__TRANSFER.toUint248(),
↪   msg.sender);
}
```

Additionally, depending on your desired functionality, you may want to add

```
|| senderRights & MultiPool.RIGHTS__PROTOCOL != 0
```

to the protocol bypass (rights protocol docs say it bypasses the check for the caller).

Kiln: Resolved by PR 155.

Spearbit: Resolved.

5.3.5 `Liquid20A` and `Liquid20C` allow sanctioned users to `transferFrom`

Severity: Low Risk

Context: Liquid20A.sol#L54-L55, Liquid20C.sol#L45-L46

Description: The `transferFrom` function does not check if the `msg.sender` is sanctioned. In the event that an address or contract is added to the sanctioned list because it is acting maliciously or on behalf of a sanctioned entity, that address could still execute `transferFrom` calls for any user's that had previously granted it an allowance.

Recommendation: Add `revertIfSanctioned(msg.sender)` to the `transferFrom` calls.

Kiln: Resolved by PR 155.

Spearbit: Resolved.

5.3.6 `sanctionsActive` is cached in memory during the call to `vNFT.act` although it is possible for it to change

Severity: Low Risk

Context: vNFTAct.driver.sol#L86, vNFT.sol#L126-L129

Description: `$sanctionsActive` is cached in memory here although it is possible in some rare cases for one (an admin) to reenter the `vNFT` and call `setSanctionsActivation` to change this storage value during the `act` sub call frame, although the `cdx` would use the old value.

Recommendation: It would be best to make sure `setSanctionsActivation` is `nonReentrant` as well.

Kiln: Resolved in PR 162 below by adding a `nonReentrant` modifier to `setSanctionsActivation`.

Spearbit: Verified.

5.3.7 `MAX_OWNERSHIP_DEPTH` should be used instead of `MAX_EXISTING_OWNERSHIP_DEPTH` when resolving group ownership

Severity: Low Risk

Context: vNFTAct.driver.sol#L216, vNFTAct.driver.sol#L373

Description: In the context, `MAX_EXISTING_OWNERSHIP_DEPTH` is used when resolving the group ownership. Although the depth of the group ownership is not changed. According to the Kiln's team:

> Only role of the MAX_EXISTING_OWNERSHIP_DEPTH is to make sure that a recipient about to receive a group is not exceeding the maximum depth.

It only makes sense to use `MAX_EXISTING_OWNERSHIP_DEPTH` in:

- `_actionCreateGroup`

- `_actionTransferGroup`

Since in those actions the depth of the group ownership gets increased by `1`.

Recommendation: Make sure in the context provided above, one is using `MAX_OWNERSHIP_DEPTH`:

- vNFTAct.driver.sol#L216 `_purchase`.
- vNFTAct.driver.sol#L373 `_actionTransfer`.

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.3.8 `_actionTransferGroup` allows for bypassing the block in `_transfer` for exited validators

Severity: Low Risk

Context: vNFTAct.driver.sol#L498

Description: A malicious user could transfer an `exited` validator despite the check in `_transfer` using groups. The user could transfer their token to a group they own, call `exit` on that validator, wait and claim and then transfer the group.

Recommendation: It could be possible to require a `tokenIds` array in `_actionTransferGroup` that checks whether they have been exited and prevent the transfer in that case; however, that might create a depth problem and/or a bad UX for users because they would have to move validators out of a group before exiting them if they were not going to exit all validators in the group. At a minimum any integrations that allow transferring groups should ensure users are aware that the validators owned by them could be exited and provide tooling that shows them which, if any, have been before they rely on groups for sales, staking, etc...

Other possible solutions to investigate:

- Should groups be represented by a hash of their contents such that when those contents change (i.e. transfer or exit) it is obvious to anyone interacting with the group.
- To solve this specific exit issue, would it make sense to have exited validators that are owned by a group revert to being owned by the address that owns the group so that they are no longer part of the group. Investigation needs to be done to ensure that that group owner who would then own the exited validator would still be able to retrieve awards.

Kiln: Resolved by PR 161. Exiting a validator transfers it to the resolved owner if `rawOwner != resolvedOwner`. The transfer does not perform the usual autoclaim to prevent an extra external call in the flow.

Spearbit: By removing exited validators from group ownership, this resolves this issue.

5.3.9 Admin or authoriser can increase nonce of any account

Severity: Low Risk

Context: vNFT.sol#L200-L202

Description: The access protected `incrementNonce` function takes an `address account` as input and increases its nonce. Using this function admin can increase nonce of authorizer and authorizer can increase nonce of admin. They both can increase nonce of any other account.

Recommendation: Consider changing the implementation to this:

```
function incrementNonce() external nonReentrant onlyAdminOrAuthorizer {
    _incrementNonce(msg.sender);
}
```

Or at least make sure the `authoriser` cannot increment the nonce of the `admin`.

Kiln: Acknowledged as the ability to increment the nonce of another account is a security feature to be able to revoke an off chain authorizations for an account by making the next nonce useless.

Spearbit: Acknowledged.

5.3.10 Unnecessary overwriting of `$mintCounter` in `vNFTActDriver`

Severity: Low Risk

Context: vNFTAct.driver.sol#L233

Description: The `_purchase` function unnecessarily overwrites the `$mintCounter` state with the same value. As the `_purchase` function calls the `_mint` function which already correctly updates the `$mintCounter` state, there is no need to update that state again in `_purchase`.

Recommendation: Consider removing the unnecessary statement from `_purchase`.

```
- $mintCounter.set(mintCounter + tokenIds.length);
```

Kiln: Resolved by PR 162.

Spearbit: Verified.

5.3.11 `MinimalRecipient.exec` function should be marked as `payable`

Severity: Low Risk

Context: MinimalRecipient.sol#L58

Description: The `MinimalRecipient` contracts are designed to be the immutable contracts on which the rewards and withdrawal amounts of validators are received. The `vFactory` uses the `MinimalRecipient.exec` function to manage and retrieve those funds.

As the `vFactory` is an upgradable contract, in future a need may arise to call the `MinimalRecipient.exec` function with some ETH value. In that case the `exec` function will revert.

Recommendation: Consider making the `MinimalRecipient.exec` function `payable` to maximize its future compatibility.

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.3.12 The `NFT` contract does not completely follow the `ERC165` standard

Severity: Low Risk

Context: NFT.sol#L140-L142

Description: The `NFT.supportsInterface` function intends to follow the Standard Interface Detection spec ERC165.

However the standard has a condition which says that the `supportsInterface` function must return `true` for `0x01ffc9a7` as the `interfaceID` (EIP165 interface). This condition is not satisfied by the `NFT.supportsInterface`.

This issue can break the integration of NFT contract with external contracts which assumes it to be fully ERC165 compliant. The external contracts will first be making the `NFT.supportsInterface(0x01ffc9a7)` call which will incorrectly return `false`.

The `vNFT` and `vExitQueue` contracts are being impacted by this bug.

Recommendation: Consider changing the implementation

```
  function supportsInterface(bytes4 interfaceId) external pure returns (bool) {
-    return interfaceId == type(IERC721).interfaceId || interfaceId ==
↪  type(IERC721Metadata).interfaceId;
+    return interfaceId == type(IERC721).interfaceId || interfaceId ==
↪  type(IERC721Metadata).interfaceId || interfaceId == type(IERC165).interfaceId;
  }
```

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.3.13 `vNFTAct.driver._pushMorphed` is incorrect

Severity: Low Risk

Context: vNFTAct.driver.sol#L693-L703

Description: This function uses the `ctx.minted` array in a couple of places and pushes the new morphed tokens into the `minted` context. Since `morphedLength` will always be 0, if `ctx.minted.length > 0` there will be `array-out-of-bounds` errors. If `tokenIdsToCopy` and `ctx.minted.length > 0`, the minted token Ids would be lost.

Recommendation: Corrected function:

```
function _pushMorphed(ActContext memory ctx, uint256[] memory tokenIds, uint256 tokenIdsToCopy)
↪  internal pure {
    uint256 morphedLength = ctx.morphed.length;
    uint256[] memory newMorphed = new uint256[](morphedLength + tokenIdsToCopy);
    for (uint256 idx = 0; idx < morphedLength; ++idx) {
       newMorphed[idx] = ctx.morphed[idx];
    }
    for (uint256 idx = 0; idx < tokenIdsToCopy; ++idx) {
        newMorphed[morphedLength + idx] = tokenIds[idx];
    }
    ctx.morphed = newMorphed;
}
```

Kiln: Resolved by PR 160.

Spearbit: Correction implemented.

5.3.14 `vFactory.depositFromRoot` changes to MinimalRecipient implementation could result in loss of funds

Severity: Low Risk

Context: vFactory.sol#L271-L273

Description: For Withdraw channel 0 deposits, the `vFactory` relies on deterministically calculating the withdraw address based on cloning an implementation, ultimately a `CREATE2` call which is determined by the caller, a salt and the bytecode of the contract being deployed.

The `MinimalRecipient` and the `vFactory` contract are upgradeable, thus their implementation can be changed in the future. If this occurs, it will be necessary to change the `vFactory`'s reliance on `_getWithdrawalAddressFromRawSalt` to retrieve withdraw addresses to pull addresses from state or use a `_getOldWithdrawalAddressFromRawSalt` function that references the previous (probably versioned) implementation to deterministically predict the address.

This migration, if implemented correctly, should work for contracts that are previously deployed by the `vFactory` (i.e. `Channel=0 && gweiThreshold > 0` or Channel=0 with a previous `_withdraw`).

However, when `depositFromRoot` is called for a deposit on withdraw channel 0 and the `gweiThreshold` is set to 0, the deposit does not deploy a new contract. It predicts the address where that contract would be deployed and

uses that on the assumption that when it comes time to withdraw, it will be able to re-predict that address and deploy the contract if needed.

If an update to the implementation occurs between the time users deposit with those conditions and attempt a withdraw, the current code could not deploy contracts at the previously predicted address because the implementation and thus the bytecode for `CREATE2` will have changed. These contracts could not be deployed by other means since `CREATE2` also takes the sender's address into account in determining the address. Meaning withdraws will fail and/or funds will not be retrievable from those addresses.

Recommendation: As upgrades may occur in less than ideal circumstances, one recommendation would be to put some work and planning now into how you would upgrade the MinimalRecipient implementation and what changes would be required in the `vFactory` to make such and upgrade successful.

Robust testing around the "*happy paths*" (such as `Channel=0` and `gweiThreshold=0`) that ensure all expected functionality works successfully will help with the current implementation as well as provide a framework with which to test upgrades.

Lastly, testing and considering options such as having `_getWithdrawalAddressFromRawSalt` and `_deployWithdrawalAddressFromRawSaltAndInit` that work with previous implementations will likely be necessary to ensure no funds are locked by undeployed and now undeployable MinimalRecipients. Special consideration should be placed on evaluating the upgrade path if a critical issue with the current MinimalRecipient is the reason for that upgrade and maintaining deployed contracts with the old implementation is not possible/recommended.

Kiln: Resolved by PR 158. Storing `address dedicatedRecipient` instead of `bytes32 recipientHash`.

Spearbit: These changes look good. We would also recommend ensuring both the `withdrawalAddress` functions have unit tests to ensure no regressions in their functionality especially around the `bytess calldata publicKey` interface one since it does not appear to be as used in the tests and now has two logic branches.

5.3.15 `vNFTActDriver._actionReconcile`: Sanctioned users can reconcile

Severity: Low Risk

Context: vNFTAct.driver.sol#L598

Description: Unlike other actions implemented in the `vNFTActDriver` contract, `_actionReconcile` is missing any checks for sanctioned users (and permissions), which means sanctioned users can call this function for their token ids.

Recommendation: Consider adding a call to `_checkRightsAndSanctions` that will block sanctioned users as well as adding a right to configure reconcile.

Kiln: Fixed in in commit 37193398.

Spearbit: Fixed by implementing the auditor's recommendation.

5.4 Gas Optimization

5.4.1 Provide `bytes32` as key to `_setDriver` instead of `string`

Severity: Gas Optimization

Context: Nexus.sol#L321-L324, Nexus.sol#L176, Nexus.sol#L202-L204

Description: It would cost less gas to make `_setDriver` to take `bytes32` instead of `string` so `keccak256(bytes(key)).k()` can be computed else where outside of the `Nexus` contract.

Recommendation: It can be changed to:

```
function _setDriver(bytes32 key, address _driver) internal {
    $drivers.get()[key.k()] = _driver.v();
    emit SetDriver(key, _driver); // <--- note that the event signature would also change
}
```

Kiln 2: Acknowledged. The contract logic related to the drivers is not invoked often, and it would change the signature of an event we're monitoring.

Spearbit: Acknowledged.

5.4.2 Use the already cached stack variable in `recipientType` in `_resolveRecipient`

Severity: Gas Optimization

Context: vNFT.internals.sol#L304

Description: In the above context `recipient.recipientType()` is calculated twice. It might be cheaper to use the already declared parameter:

```
Recipient.RecipientType recipientType = recipient.recipientType();
```

Recommendation: The code could be changed to:

```
Recipient.RecipientType recipientType = recipient.recipientType();
if (recipientType == Recipient.RecipientType.Address) {
    return recipient.getAddress();
}
```

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.4.3 Use contract-level constants

Severity: Gas Optimization

Context: AccountList.sol#L141-L143, AccountList.sol#L151, Nexus.sol#L381, vNFT.internals.sol#L124

Description: Hashes computed in this context get recomputed for each call to the scope.

- AccountList.sol#L141-L143
- AccountList.sol#L151
- Nexus.sol#L381
- vNFT.internals.sol#L124

Recommendation: It would be best to define contract-level constants with their value as the computed hashes.

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.4.4 Unnecessary validation checks for `enum` datatype

Severity: Gas Optimization

Context: vNFT.sol#L404-L406

Description: Solidity's default type checking of `enum` already prevents the `enum` value to go over the defined range. Decoding of input data automatically fails when input is greater than defined `enum vNFTMode` type range.

Recommendation: Consider removing this check:

```
- if (config.mode > ctypes.vNFTMode.Dynamic) {
-     revert InvalidMode();
- }
```

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.4.5 `vNFTActDriver._checkRightsAndSanctions`: Redundant extra sanctions check

Severity: Gas Optimization

Context: vNFTAct.driver.sol#L626-L628

Description: Consider the implementation of `_checkRightsAndSanctions`:

```
function _checkRightsAndSanctions(ActContext memory ctx, address account, uint248 shouldHave, uint248
↪  shouldNotHave) internal view {
    if (ctx.sanctionChecksEnabled && account != ctx.msgSender) {
        _revertIfSanctioned(account);
    }
    uint248 rights = account != ctx.msgSender ? _getRights(account) : ctx.msgSenderRights;
    _shouldHaveRights(account, rights, shouldHave);
    _shouldNotHaveRights(account, rights, shouldNotHave);
    if (ctx.sanctionChecksEnabled && account != ctx.msgSender) {
        _revertIfSanctioned(account);
    }
}
```

As we can see, the call to `_revertIfSanctioned(account)` is being made twice while the `account` variable does not change in between the calls.

Recommendation: Consider removing the second call to `_revertIfSanctioned`.

Kiln: Fixed in commit 37193398.

Spearbit: Fixed by implementing the auditor's recommendation.

5.4.6 `vNFTActDriver._actionTransfer`: Gas optimizations

Severity: Gas Optimization

Context: vNFTAct.driver.sol#L361

Description:

1. The check of `_tokenRawOwner(tokenId) != from` is redundant as it is already part of `_transfer`.
2. The check of `to == bytes32(0)` is redundant as it is already part of `_transfer`.
3. `tokenIds[transferIdx]` is being used twice (redundant `MLOAD`), `tokenId` can be used in the second time instead.
4. `ctx` and `transferParameters` do not change in the for loop. Resolving and checking the rights and sanctions can happen before the loop.

Recommendation: Consider replacing the function with this code snippet that includes all the changes mentioned above.

```
function _actionTransfer(ActContext memory ctx, Action calldata action, uint256[] memory tokenIds)
↪  internal {
    uint256 tokenIdsLength = tokenIds.length;
    if (tokenIdsLength == 0) {
        revert EmptyTokenArray();
    }

    if (!_isApprovedOrTokensOwner(ctx.msgSender, tokenIds)) {
        revert LibErrors.Unauthorized(ctx.msgSender, address(0));
    }

    TransferParameters memory transferParameters = abi.decode(action.data, (TransferParameters));
    (bytes32 from, address fromOwner) = _resolveAll(ctx, transferParameters.from, MAX_OWNERSHIP_DEPTH);
    (bytes32   to, address   toOwner) = _resolveAll(ctx, transferParameters.to  , MAX_OWNERSHIP_DEPTH);

    _checkRightsAndSanctions(ctx, fromOwner, RIGHTS__TRANSFER, RIGHTS__FORBIDDEN);
    _checkRightsAndSanctions(ctx,   toOwner, RIGHTS__TRANSFER, RIGHTS__FORBIDDEN);

    for (uint256 transferIdx = 0; transferIdx < tokenIdsLength; ++transferIdx) {
        _transfer(from, to, toOwner, tokenIds[transferIdx]);
    }
}
```

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.4.7 `vNFTActDriver._purchase`: Gas optimizations

Severity: Gas Optimization

Context: vNFTAct.driver.sol#L149, vNFTAct.driver.sol#L200-L202

Description:

1. vNFTAct.driver.sol#L149 assigns value to `toRights`. in this case the check of `from != to` is not needed since `from` will always be different from `to` in this code path.

```
if (from == to) {
  // ...
} else {
  // ...
    uint248 toRights = from != to ? _getRights(to) : fromRights;
    _shouldHaveRights(to, toRights, RIGHTS__TRANSFER);
    _shouldNotHaveRights(to, toRights, RIGHTS__FORBIDDEN);
}
 // ...
```

2. vNFTAct.driver.sol#L200-L202 compares the `tokenIds` and `validationKeys` lengths and is also redundant since this check is already being made inside the `depositFromRoot` function:

```
// ...
if (tokenIds.length != validationKeysLength) {
    revert InvalidPurchasedValidatorCount($$.validationKeys.length, tokenIds.length);
}
// ...
```

   - `src/vFactory.sol`:

```
    `function depositFromRoot(DepositFromRootParameters calldata $$) external payable
↪   onlyDepositor($$.wc) returns (uint256[] memory) {
        // ...
        uint256 validationKeysLength = $$.validationKeys.length;
        // ...
        {
            // ...
            __.ids = new uint256[](validationKeysLength);
           // ...
        }
        // ...
        return __.ids;
    }
```

Kiln:

- (1) Is fixed in PR 161.
- (3) Is not fixed since it's something that was raised in our previous audit where client did a similar thing at the `vPool` level when calling `depositFromRoot`. Both checks will be kept.

Spearbit: Verified.

5.5 Informational

5.5.1 Override `_exists` in `vNFT.internals` to check non-emptiness of ownership without truncating

Severity: Informational

Context: NFT.sol#L393, NFT.internals.sol#L249, vNFT.sol#L334-L337, NFT.sol#L250, NFT.sol#L256, NFT.sol#L377

Description: Currently, `vNFT.internals` inherits `_exists` from `NFT` which upon checking the ownership of the token truncates the `$owners.get()[tokenId]` which might loose some upper bits related to some special flags such as group ownership in this current implementation. Currently this is not an issue since group raw ids start from `0x1100000000000000000000000000000000000000000000 + 1` and thus truncation for the GROUP ownership space would also result on a non-zero value.

Note that `_exists` or it's counterpart implementation is used in:

1. `_mint(address to, uint256 tokenId)`, before and after the hook call `_onMint` to make sure the token id is not minted yet.
2. `_mint(bytes32 staker, bytes32 to, uint256 tokenId)` to make sure the token id is not minted yet.
3. `_requireExists` to make sure the token id is minted and thus have an owner.
4. `tokenURI` used the alternative and more current overwritten implementation of `_requireExists` given by:

```
address tokenOwner = _tokenResolvedOwner(tokenId, MAX_OWNERSHIP_DEPTH);
if (tokenOwner == address(0)) {
    revert InvalidTokenId(tokenId);
}
```

Recommendation: It is safer to avoid the truncation of token id ownership in `_exists` using the following implementation which is equivalent to the above cases in the current implementation due to that facts that currently there is only one more ownership space for group and group raw ids start from a non-zero value:

```
// in vNFT.internals

function _exists(uint256 tokenId) internal view override(NFT) returns (bool) {
    return _tokenResolvedOwner(tokenId, MAX_OWNERSHIP_DEPTH) != address(0);
}
```

and then `tokenURI` can also be updated to:

```
function tokenURI(uint256 tokenId) external view override(IERC721Metadata, NFT) returns (string memory)
↪  {
    _requireExists(tokenId); // <--- changed code
    string memory metadataURIBase = $configuration.get().metadataURIBase;
    return string(abi.encodePacked(metadataURIBase, Strings.toHexString(address(this)), "/",
↪  Strings.toString(tokenId)));
}
```

Kiln: Fixed in commit f433be75.

Spearbit: Fixed in commit f433be75. by implementing the auditor's recommendation.

5.5.2 Zero memory slot is used when computing

Severity: Informational

Context: array.sol#L26-L38, array.sol#L37

Description: The code in this context is:

```
mstore(0x60, position)
let startPtr := keccak256(0x60, 0x20)

// <LOOP_BLOCK>

// Clean up the scratch space
mstore(0x60, 0)
```

which used to be:

```
mstore(0, position)
let startPtr := keccak256(0, 0x20)

// <LOOP_BLOCK>
```

So before the change, the codebase was using the first scratch space slot, but no it is using the zero memory slot which needs to reset back to `0`.

1. The initial implementation was safe.
2. In the current implementation the following comment is inaccurate:

   ```
   // Clean up the scratch space
   ```

3. One can reset the zero memory slot right after the hashing operation.

Recommendation: Either use the original implementation or apply the following changes:

```
mstore(0x60, position)
let startPtr := keccak256(0x60, 0x20)

// reset the zero memory slot to `0`
// here we are resetting the `0x60` memory slot right after the hashing operation
mstore(0x60, 0)

// <LOOP_BLOCK>
```

Kiln: Fixed in PR 155.

Spearbit: Verified.

5.5.3 Rebates are not applied to validators who exit volunteerly

Severity: Informational

Context: vNFT.sol#L219-L229, vNFTAct.driver.sol#L453-L454

Description: If a validator exists without listening to the event emitted by the `vFactory` which should come from the exit action of `vNFT`, the `32 ETH` rebates would not be applied for the `validatorOwner` and the protocol would take commission on that amount.

Recommendation: The above should be documented and monitored to apply correction in case such a scenario occurs.

Kiln: Acknowledged. This is similar to what we're already doing in the v1 of our product on LL.

Spearbit: Acknowledged.

5.5.4 Possible Invariants to test

Severity: Informational

Context: vNFT.internals.sol#L176-L177

Description/Recommendation: Some possible invariants to add:

- Burn counter is always less than or equal to Mint Counter.
- Neither counter can decrease.
- The sum of user balances = `$mintCounter` - `$burnCounter` (`totalSupply`).

Kiln: Acknowledged. We are planning a revamp specifically for invariant tests around integration contracts and this will be covered.

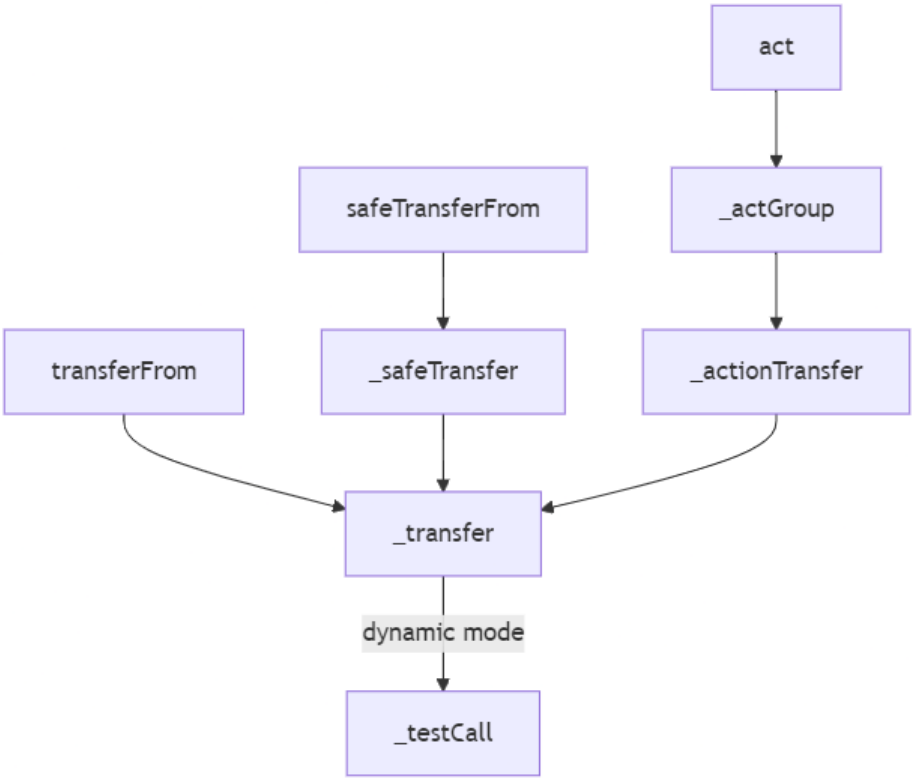
Spearbit: Acknowledged.

5.5.5 Move the block that calls `_testCall` to the end of `_transfer` function

Severity: Informational

Context: vNFT.internals.sol#L209-L214, vNFT.internals.sol#L235-L238

Description: We have the following call function flow graph:



In `_transfer`, `_testCall` is called if `vNFT` is configured to be in a `Dynamic` mode. After this call some checks and state updates happen:

```
if (_tokenRawOwner(tokenId) != from) {
    revert InvalidFrom(from, _tokenRawOwner(tokenId));
}

// Clear approvals from the previous owner
delete $tokenApprovals.get()[tokenId];

unchecked {
    $balances.get()[from.k()] -= 1;
    $balances.get()[to.k()] += 1;
}
$owners.get()[tokenId] = to.v();
```

also before the call some checks and state updates are performed.

Recommendation: It would be best to move the block that calls `_testCall` to the end of `_transfer` function so that all the checks and state updates happen before this stateful call to an external contract:

```
// ...

if (_tokenRawOwner(tokenId) != from) {
    revert InvalidFrom(from, _tokenRawOwner(tokenId));
}

// Clear approvals from the previous owner
delete $tokenApprovals.get()[tokenId];

unchecked {
    $balances.get()[from.k()] -= 1;
    $balances.get()[to.k()] += 1;
}
$owners.get()[tokenId] = to.v();

_transferEvent(from, to, tokenId);

if ($configuration.get().mode == ctypes.vNFTMode.Dynamic) {
    _claimSingle(validator.id, false);
    if (toOwner.code.length > 0 && !_testCall(toOwner)) {
        revert RecipientCannotReceiveRewards(to);
    }
}
```

Also since you might be able to `call` a contract but might not be able to sent any native tokens or there might be a cap on how much it can receive, this stateful call in `_testCall` does not guarantee that the contract can receive native tokens and only adds a potential external call point in the transfer flow. It might be best to remove it altogether:

```
if ($configuration.get().mode == ctypes.vNFTMode.Dynamic) {
    _claimSingle(validator.id, false);
   // check whether `toOwner` supports an interface when it is a contract
    if (toOwner.code.length > 0 && IERC165(toOwner).supportInterface(...)) {
        revert RecipientCannotReceiveRewards(to);
    }
}
```

or one can enforce that the `toOwner` should implement an interface and instead static call its `supportInterface` endpoint.

Kiln: Resolved by PR 162. Decided to remove the call.

Spearbit: Verified.

5.5.6 Typos, incorrect comments, missing documentations, ...

Severity: Informational

Context: vNFT.sol#L87, vNFT.internals.sol#L190-L194, vNFTAct.driver.sol#L114-L122, IvFactory.sol#L297, NFT.sol#L55, NFT.sol#L59, NFT.sol#L63

Description/Recommendation:

1. `onwers` → `owners`.
2. Missing full documentation of params.
3. Missing documentation about return values.
4. Incorrect comment - In most other places it is referred to as 0 channel not `null` and I don't see any reason this function is not callable on validators that have been funded and then exited (i.e. are no longer funded).

5. NFT.sol#L55, NFT.sol#L59, NFT.sol#L63, the NatSpec for `$owners`, `$balances`, and `$tokenApprovals` mentions that the type is of `mapping (uint256 => address)` or `mapping (uint256 => address)` both are incorrect since one is using `using LMapping for types.Mapping` and `LMapping.get` returns `mapping(uint256 => uint256) storage data`.

Kiln:

- (2) `toOwner` has been removed as an input parameter. Thus the NatSpec is not required anymore. This also means the empty call to `toOwner` has also been removed.

  The other tasks have been also fixed in PR 162.

- (5) has been fixed in 883cd5bb by adding an extra comment regarding the actual type.

Spearbit: Verified.

5.5.7 `vNFT.internals` Optimizations

Severity: Informational

Context: vNFT.internals.sol#L177, vNFT.internals.sol#L245-L247

Description/Recommendation:

1. Reuse `newTokenId` variable instead of doing the math again.
2. Unnecessary check of `to == bytes32(0)` in `_mint`. Due to vNFTAct.driver.sol#L206 `to` cannot be `bytes32(0)` at this point.

Kiln: Resolved by PR 162.

Spearbit: Changes look good.

5.5.8 `vCoverageRecipient` low level call error and event consistency

Severity: Informational

Context: vCoverageRecipient.sol#L152-L158

Description: In most other places where a call is made to send value the event is emitted after the success check is made and there is a custom error used.

Recommendation: Consider adding a custom error such as `RemoveEtherError(recipient, rdata)` and move the `UpdatedEtherForCoverage` event emission until after the success check is made.

Kiln: Only the custom event:

```
RemoveEtherError(recipient, amount, rdata);
```

has been introduced. The `UpdatedEtherForCoverage` event is still emitted before checking for errors.

Fixed in PR 162.

Spearbit: Verified.

5.5.9 Missing inheriting `Implementation` contract for `vNFTActDriver`

Severity: Informational

Context: vNFTAct.driver.sol#L44

Description: Similar to `vPoolReportingDriver` the `vNFTActDriver` contract should also inherit the `Implementation` contract. This way the implementation of different drivers of Kiln protocol remains consistent.

Recommendation: Consider inheriting `Implementation` contract for `vNFTActDriver`.

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.5.10 Missing `nonReentrant` modifier on some external functions of `vNFT`

Severity: Informational

Context: vNFT.sol#L126, Administrable.sol#L58, Administrable.sol#L52

Description: The `vNFT` contract contains these external functions on which the `nonReentrant` modifier is not applied:

- `setSanctionsActivation`
- ☐ `transferAdmin`
- ☐ `acceptAdmin`

The re-entrancy guard isn't needed for these functions but since it has been applied on other admin protected functions (like `setAuthorizer`) it should be applied to the mentioned three functions as well.

Recommendation: Consider applying the `nonReentrant` modifier on the mentioned functions.

Kiln: Partially resolved by PR 162. The admin methods are left without `nonReentrant` as they're in another contract and this will add the reentrancy guard on all the contracts of the system while not needed

Spearbit: Verified.

5.5.11 Unused `RIGHTS__NONE` constant variable

Severity: Informational

Context: vNFT.internals.sol#L98

Description: The `RIGHTS__NONE` constant variable is declared but never used.

Recommendation: Consider removing the unused variable.

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.5.12 Missing function to read the `$validatorIdToTokenId` state of `vNFT`

Severity: Informational

Context: vNFT.internals.sol#L82

Description: The `vNFT` contract is missing a `view` function to read the `$validatorIdToTokenId` state. A `view` function which takes `validatorId` as input and returns the NFT's `tokenId` (by simply reading `$validatorIdToTokenId`) should be exposed.

This is needed as the `purchase` function returns `validatorIds` and the `act` function accepts `tokenIds`. Considering that the `tokenIds` can be morphed, a `view` function will come handy.

Recommendation: Consider adding the `view` function.

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.5.13 Consistency of constant declarations can be improved in `Recipient`

Severity: Informational

Context: Recipient.sol#L24-L26

Description: The `GROUP_MASK`, `SPECIAL_MASK` & `INVALID_MASK` constant variables can be declared using the same format as account rights declarations. So that the declaration format remains consistent throughout the Kiln protocol.

Recommendation: Constants should be declared as:

```
uint256 internal constant GROUP_MASK = 1 << 255;
uint256 internal constant SPECIAL_MASK = GROUP_MASK >> 1;
uint256 internal constant INVALID_MASK = SPECIAL_MASK >> 1;
```

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.5.14 Empty and uninvoked hooks in `vNFTInternals`

Severity: Informational

Context: vNFT.internals.sol#L269-L279

Description: The `vNFTInternals` inherits the `NFT` contract. The `vNFTInternals` defines `_onTransfer`, `_onMint` & `_onBurn` empty hooks but doesn't trigger them while transfer or minting operations. The `_onBurn` hook gets triggered by the inherited `_burn` function but the `_burn` function is never used in the contract's lifecycle.

This behaviour of defining a hook but not triggering it or triggering only one of the defined hooks seem inconsistent. This may raise issues if `vNFTInternals` is further inherited by future contracts which assumes certain behaviour from the hooks.

Recommendation: Since Kiln does not intend to invoke the hooks it would be better if all hooks are made uninvokable explicitly.

```
function _onTransfer(address, address, uint256) internal override(NFT) {
    revert UnsupportedMethod();
}
```

Same for other hooks.

Kiln: Acknowledged. By adding a revert in the hooks, we then get tens of warnings due to unreachable code in `NFT.sol`. Example:

```
Warning (5740): Unreachable code.
   --> src/utils/NFT.sol:325:9:
    |
325 |         emit Transfer(from, to, tokenId);
    |         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

We prefer keeping our CI as is and fail whenever we have a compilation warning.

Spearbit: Verified.

5.5.15 Optimization of `vNFTInternals._transferEvent` function

Severity: Informational

Context: vNFT.internals.sol#L123-L128

Description: The `_transferEvent` functions has a few areas of improvement:

1. The function converts the `keccak256` hash to `bytes32` which is not required. The output of `keccak256` is already `bytes32`.
2. On every invocation the function computes `keccak256` hash of string `Transfer(address,address,uint256)` which is unnecessary and cost additional gas. Rather the hash value can be stored as a contract level constant.

Recommendation: Consider changing the implementation to:

```
bytes32 internal constant TOPIC_TRANSFER = keccak256("Transfer(address,address,uint256)");
// ...

function _transferEvent(bytes32 from, bytes32 to, uint256 tokenId) internal {
    bytes32 topic = TOPIC_TRANSFER;
    assembly {
        log4(0, 0, topic, from, to, tokenId)
    }
}
```

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.5.16 The statements of `vFactory._setTreasury` can be inlined into the `initialize` function

Severity: Informational

Context: vFactory.sol#L622-L626

Description: In `vFactory` the internal `_setTreasury` function is only called once in the `initialize` function. Its statements can be inlined into the `initialize` function itself.

Recommendation: Move the statements of `_setTreasury` function into the `initialize` function.

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.5.17 Redundant function `driverCallWithValue` is present in `LibDriver` library

Severity: Informational

Context: LibDriver.sol#L49

Description: The `driverCallWithValue` function in `LibDriver` library contains the exact same code and logic as the `driverCall` function. In EVM there is no need to pass `value` when performing a `delegatecall`.

Recommendation: Consider removing the `driverCallWithValue` function.

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.5.18 Internal `_incrementNonce` function should be used to increase the nonce value

Severity: Informational

Context: AccountList.sol#L135

Description: The `AccountList._verifyApplyRightsAuthorization` function directly writes to `$nonces` state to increase the nonce value. Instead it should use the internal `_incrementNonce` function for better code consistency.

Recommendation: Consider using the `_incrementNonce` function to increase the nonce value.

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.5.19 Missing natspec documentation for `vNFTMode`, `vNFTConfiguration` and `vNFTValidator`

Severity: Informational

Context: ctypes.sol#L72-L88

Description: Missing natspec documentation for `vNFTMode`, `vNFTConfiguration` and `vNFTValidator` types in `ctypes.sol`.

Recommendation: Consider adding appropriate natspec docs for the specified types.

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.5.20 `vNFTAct.driver` optimizations

Severity: Informational

Context: vNFTAct.driver.sol#L249-L250, vNFTAct.driver.sol#L285-L288, vNFTAct.driver.sol#L398-L401, vNFTAct.driver.sol#L419, vNFTAct.driver.sol#L512-L518, vNFTAct.driver.sol#L683

Description/Recommendation:

1. This loop could be simplified and brought inline with the `minted` loop below by removing `tokenId` declaration and using `tokenIds[idx] = ctx.morphed[idx];`.
2. `_actionExit` already fetches `$validators.get()` from storage so it can check their exited state. `_resolveTokenIds` is unnecessary here as it could be constructed with fresh data in the exit action. This would have the added benefit of avoiding stale `validatorId` data from looped actions or groups.
3. `tokenIdsLength` is only used in this if statement so can just use `tokenIds.length` directly.
4. Can be `unchecked{ ++morphedIdx; }` for slight optimization.
5. You could probably simplify your interfaces and event monitoring by removing the `IllegalTransferToZero` and using `InvalidTransfer(from, bytes32(0))`.

6. Use length variable instead of `.length` again.

Kiln: Points 1 to 5 Resolved by PR 162. Point 6 Resolved by PR 160.

Spearbit: Resolved.

5.5.21 `vNFT.driver` unnecessary recipient checks

Severity: Informational

Context: vNFTAct.driver.sol#L208-L210, vNFTAct.driver.sol#L206-L219

Description: Based on the current `RecipientTypes`, the `InvalidRecipient error` will never be hit. vNFTAct.driver.sol#L206 sets this to `$$.recipient` if there is one, but `$$.recipient` is already checked for being either `Address` or `Group` when it gets resolved to the `to` address above it.

Recommendation: Remove unnecessary check/revert. Further this block (second context), could be bypassed for some actions. If `$$._toRecipient == bytes32(0) && toRecipient == fromRecipient` then there is no need to check the types or groups (`fromRecipient` has to be type Address since it is originally set as `msg.sender`).

Kiln: Resolved by PR 162.

Spearbit: Resolved.

5.5.22 `vNFT.internals` unnecessary lock/unlock functions

Severity: Informational

Context: vNFT.internals.sol#L180-L188

Description: These functions are only used in one place (`_claim`).

Recommendation: Remove the internal functions and use `$locked.set` directly in `_claim`.

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.5.23 `FeeDispatcher` Optimizations

Severity: Informational

Context: FeeDispatcher.sol#L81, FeeDispatcher.sol#L93-L98, FeeDispatcher.sol#L118

Description:

1. `++i` is more efficient.
2. In most other places where a `call` is made and check there is a success check with a custom error and then the success event is emitted.
3. Unless there is a specific need for this error and the index, this could be performed with `LibSanitize.nonNullValue(split)`

Recommendation:

1. Use:

```
   unchecked {
-      i++;
+      ++i;
   }
```

2. Consider changing to:

```
(bool success, bytes memory rdata) = recipient.call{value: amount}("");
if (!success) {
   revert SendCommissionError(recipient, rdata);
}
emit CommissionWithdrawn(recipient, amount);
```

3. Consider re-using `LibSanitize.nonNullValue(split)`.

Kiln: Resolved by PR 155.

Spearbit: Verified.

5.5.24 `ISanctionsList` unnecessary interface

Severity: Informational

Context: ISanctionsList.sol#L16-L18

Description/Recommendation: This function interface does not appear to be used and can be removed.

Kiln: Resolved by PR 162.

Spearbit: Looks good.

5.5.25 `Depositor._deposit` allows sending too much ETH which will be used/taken by the next depositor

Severity: Informational

Context: Depositor.sol#L69

Description: Depositors interacting directly with `vFactory.depositFromRoot` could lose their ETH if they send more than needed (i.e. `32 ETH * validationKeys`).

Because the deposit contract sweeps whatever ETH into its deposit if a depositor does this, then next depositor can grab the leftover ETH to deposit it (+ whatever is needed to make 32 ETH total) and then withdraw it.

This is most likely a UX issue as a user should not send more than the required amount.

Recommendation: The deposit contract could either revert if it receives an incorrect amount, possibly the extra or send it along as though it were rewards to be claimed.

Kiln: Acknowledged, but won't change since 32 ETH may not always be the deposit amount.

Spearbit: Acknowledged.

5.5.26 Consider documenting the behavior of `FeeDispatcher` in parent contracts

Severity: Informational

Context: FeeDispatcher.sol#L73

Description: The `vNFT` contract inherits the `FeeDispatcher` contract which sends the entire eth balance of the contract upon a call to `withdrawCommission`. It is important to mention that we could not find any issues in the current version of the code, however we do think that this type of design may increase the surface area due to the fact that the contracts are upgradeable. Developers of future versions may not be fully aware of this implicit assumption and may accidentally design features that are supposed to leave user funds in the contract which can be later "stolen" by fee recipients.

Recommendation: Consider documenting this behavior of the `FeeDispatcher` in all parent contracts as well as in internal docs meant for developers/integrators.

Kiln: Fixed in PR 162.

Spearbit: Verified.

5.5.27 Consider variables and functions renamings

Severity: Informational

Context: vNFTAct.driver.sol#L174, vNFTAct.driver.sol#L675, vNFTAct.driver.sol#L502, FeeDispatcher.sol#L91

Description/Recommendation:

- `vNFTAct.driver._purchase`: `tokenIds` in line 174 should be renamed to `validatorIds` to avoid any confusion with the `vNFT` token ids introduced later in the function.
- ☐ `vNFTAct.driver._pushMinted`: `ctx.minted` should be renamed to `ctx.tokens` and `ctx.groups` should be renamed `ctx.mintedGroups`.
- ☐ `vNFTAct.driver._actionTransferGroup`: `_resolveRecipientFromContext` should be changed to a name that will also reflect this case of function invocation in which the resolved `group` is not a recipient.

  FeeDispatcher.sol#L91, `sendCommissionToRecipient` should be renamed to `_sendCommissionToRecipient` since it is an internal function.

Kiln: Partially Resolved by PR 162 (only 1 and 4). 2 and 3 are acknowledged.

Spearbit: Verified.

5.5.28 `vNFT.safeTransferFrom, transferFrom`: Consider avoiding duplicated code

Severity: Informational

Context: vNFT.sol#L285-L297

Description: Both `safeTransferFrom` functions are almost identical to `transferFrom` (besides the callback part). Duplicated code, or code that is copied and pasted multiple times within a project or across projects, is generally not a good practice in software development. It can lead to several issues, including increased maintenance costs, decreased code readability, and a higher likelihood of introducing bugs into the codebase.

Recommendation: Consider changing `safeTransferFrom` so it will call `transferFrom` instead of duplicating code, and add the additional callback after the call to `transferFrom` is returned.

Kiln: Refactoring done in PR 162.

Spearbit: Verified.

5.5.29 `vNFTActDriver.purchase` Lacks handling of potential eth residue

Severity: Informational

Context: vNFTAct.driver.sol#L69

Description: Both `purchase` and `_actionPurchase` use `_purchase` during their execution. `_purchase` is calculating the amount of eth left from the `msgValue` after purchasing and it is stored in the return value `updatedMsgValue`. the purpose of handling left over eth is to allow multiple actions in a single transaction. By the end of the call to `act` the left over amount of eth is sent back to the caller. However, this is not the case for the `purchase` flow which means any residue eth left from a call to `purchase` will stay in the contract and will be considered as fee and will be later claimed by the fee recipients. It is important to mention that since `purchase` is supposed to enable a single purchase operation then it is safe to assume that the caller would send the exact amount therefore we decided to label this issue as informational.

Recommendation: To mitigate this potential risk consider adding a check to the `purchase` function that will revert in case `msg.value` is not a multiple of 32.

Kiln: Resolved by PR 162. Decided to revert on all cases of invalid message value that can be caught, no more refund at the end.

Spearbit: Verified.