Vsuite PR 175 Security Review

Auditors

Emanuele Ricci, Lead Security Researcher

Optimum, Lead Security Researcher

Akshay Srivastav, Security Researcher

Report prepared by: Lucas Goiriz

February 18, 2025

Contents

1 About Spearbit **2**

2 Introduction **2**

3 Risk classification **2**
3.1 Impact . . . 2
3.2 Likelihood . . . 2
3.3 Action required for severity levels . . . 2

4 Executive Summary **3**

5 Findings **4**
5.1 Low Risk . . . 4
5.1.1 Additional integrity checks . . . 4
5.2 Informational . . . 6
5.2.1 Track unstructured storage changes across upgrades and migrations . . . 6
5.2.2 Wrong code comment about the new data structure for `$deposits` . . . 6
5.2.3 Old channel state might not be deleted completely . . . 7
5.2.4 `pkr` values might be overwritten in case of duplicated public keys . . . 8
5.2.5 `Migration_2_1_base` refactoring . . . 8
5.2.6 Natspec and dev comments inaccuracies in `vFactory.storage_migration` . . . 9
5.2.7 `vFactory.storage_migration` refactoring and improvements . . . 9
5.2.8 Migration specific storage states will continue to exist in `vFactory` after migration . . . 9
5.2.9 Input validation of `vFactory_2_1_storage_migration.migrate` function can be improved . . . 10
5.2.10 Non-critical issues . . . 10
5.2.11 Potential re-entrancy in `Cub._fix` operation . . . 11

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 PR 175 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 4 days in total, Kiln engaged with Spearbit to review the vsuite-PR175 protocol. In this period of time a total of **12** issues were found.

Summary

Project Name	Kiln
Repository	vsuite-PR175
Commit	af82a04d
Type of Project	Vaults, Staking
Audit Timeline	Jan 25th to Jan 29th

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	1	1	0
Gas Optimizations	0	0	0
Informational	11	9	2
Total	**12**	**10**	**2**

5 Findings

5.1 Low Risk

5.1.1 Additional integrity checks

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The `migrate_vFactory` function is responsible to migrating every `vFactory` from version `v2.0` to version `v2.2` and performing the cleaning and migration from the old data structure to the new one. While some checks are implicitly performed by the `.migrate()` function performed by `vFactory.storage_migration.sol`, in our opinion the test (and then deployment) script should treat the migration function as an untrusted entity. As a consequence, we believe that Kiln should require the `migrate_vFactories` function to ensure, without trusting any other parties, that the changes requested have been executed, and the needed events have been emitted.

Below are the changes made to the unstructured storage:

- Added the unstructured storage `mapping(address => uint256) $publicKeyRegistry`.
- Added the unstructured storage `mapping(address => uint256) $fundedValidators`.
- Changed the unstructured storage data structure of `mapping(uint256 => ctypes.Deposit) $deposits`.
- Removed `mapping(bytes32 => ctypes.WithdrawalChannel) $withdrawalChannels`.
- Removed `mapping(bytes32 => mapping(address => uint256)) $balances`.
- Removed `address $operator`.

Depending on the change type (added, changed, removed), the action and required-post-migration-test changes:

- For `$publicKeyRegistry` we must ensure that.
- Each public key registered is unique.
- Every public associated to a funded validator registered in the old `$deposits` has been migrated (`$deposits[validatorId].index - 1` represents the index of the validator in the `$withdrawalChannels.get()[$deposits[validatorId].wc].validators` array of validators associated to the wc).
- For `$fundedValidators` we must ensure that for each existing `withdrawal_channel` the value of `$withdrawalChannels.get()[withdrawal_channel].funded` has been migrated to `$fundedValidators.get()[withdrawal_channel]`.
- For `$deposits` we must ensure that every record has been correctly migrated from the old data structure to the new one.
- `dedicatedRecipient` must be equal to `address(0)`.
- `owner` must be equal to the "old" `owner` (the `vPool`).
- `feeRecipient` must be equal to fee recipient associated to the withdrawal channel associated to the validator of the deposit `$withdrawalChannels.get()[$deposits[validatorId].wc].feeRecipient`.
- `threshold` must be equal to `0`.
- For `$withdrawalChannels` we must ensure that each item of the mapping has been cleaned after the values have been migrated to the new data structure where needed.
- `lastEdit` set to `0`.
- `limit` set to `0`.
- `funded` set to `0`.
- `validators` clean item of the array of `Validator`.

- For `$operator` we must ensure that is set to `address(0)`.

In addition to these data migrations and cleaning, the `migrate_vFactory` should ensure that for every `$deposits`, the following events, with the correct and corresponding event's values, are re-triggered to emulate a `deposit` in the `vFactory` v2.2:

- `event ActivatedValidator(bytes32 indexed withdrawalChannel, address indexed depositor, address indexed withdrawalAddress, bytes publicKey, bytes signature, uint256 id).`
- `event SetValidatorOwner(uint256 indexed id, address owner).`
- `event SetValidatorFeeRecipient(uint256 indexed id, address feeRecipient).`

Note that the execution of `vFactory` v2.2 the `event SetValidatorExtraData(uint256 indexed id, string extraData)` would also be emitted. Unfortunately, it's not possible to re-emit such an event in this case, given that the `extraData` is **not** stored in the contract's state.

By following the `migrate_vFactory` flow, we can see that this migration's integrity checks are missing:

- Migration.base.t.sol#L1176: `.migrate(...)` should ensure that the `ActivatedValidator`, `SetValidatorOwner` and `SetValidatorFeeRecipient` has been emitted for each deposit with the correct values.
- For each of the deposits migrated by .migrate(...) the public key of the corresponding validator (a deposit record corresponds to a validator being funded) should have been registered in the new unstructured storage `$publicKeyRegistry`.
- Migration.base.t.sol#L1183-L1185: should ensure that the new `$deposits` record has the `threshold` attribute equal to `0`.
- After the first step of the migration (`eraseMode == false`) has been applied Migration.base.t.sol#L1189 the function should validate that.
- the `$operator` unstructured storage has been reset to `address(0)`.
- every `$balances` item tracked by the `deletionParameters` array has been correctly reset to `0`.
- After the second step of the migration (`eraseMode == true`) the logic should verify that.
- For every item of `$withdrawalChannels.get()[withdrawal_channel]` the `funded` attribute (how many validators of the withdrawal channel has been funded) has been correctly migrated to corresponding item in the new unstructured storage `$fundedValidators.get()[uint256(wc)]`.
- Every attribute of `$withdrawalChannels.get()[withdrawal_channel]` has been fully cleaned.
- `lastEdit` must be equal to `0`.
- `limit` must be equal to `0`.
- `funded` must be equal to `0`.
- `validators` (which is an array of `Validator`) must be fully cleaned. This part of the validation should have been already covered by the logic in Migration.base.t.sol#L1236-L1241.

Both `migrate_vFactories` and `vFactory_2_1_storage_migration.migrate` should ensure that the old `vFactory` did not contain duplicates of a validator's public key (see also the finding `pkr` values might be overwritten in case of duplicated public keys).

Recommendations: Kiln should implement all the suggested integrity checks that are currently missing in the `migrate_vFactories` logic. As already mentioned, the testing script should never trust that the migration function correctly apply the requested changes and validations internally.

Kiln could also consider refactoring the `migrate_vFactories` logic to cache all the needed "old" values to be able to perform these integrity checks only at the very end of the execution, where part of the data migration integrity is already performed (see Migration.base.t.sol#L1270-L1292).

Kiln: Fixed the main points in commit 54d38413. Cannot go deeper as the stack is becoming an issue for the complexity of some of the checks

Spearbit: Fix verified.

5.2 Informational

5.2.1 Track unstructured storage changes across upgrades and migrations

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The deployment of `vsuite v2.2` and the execution of the `vsuite 2.1` migration scripts will perform the following changes to the `vFactory` contract unstructured storage:

- Added `mapping(address => uint256) $publicKeyRegistry` with storage slot `keccak256(bytes("factory.1.publicKeyRegistry")) - 1`.
- Added `mapping(address => uint256) $fundedValidators` with storage slot `keccak256(bytes("factory.1.fundedValidators")) - 1`.
- Changed `mapping(uint256 => ctypes.Deposit) $deposits` struct from storage slot `keccak256(bytes("factory.1.deposits")) - 1` to `keccak256(bytes("factory.2.deposits")) - 1`.
- Removed `mapping(bytes32 => ctypes.WithdrawalChannel) $withdrawalChannels` with storage slot `keccak256(bytes("factory.1.withdrawalChannels")) - 1`. The migration takes case to also "reset" the existing values.
- Removed `mapping(bytes32 => mapping(address => uint256)) $balances` with storage slot `keccak256(bytes("factory.1.balances")) - 1`. The migration takes case to also "reset" the existing values.

The above changes are not tracked anywhere in the code or in the documentation, and it's only possible to visually track the changes by looking at the integer number that should theoretically identify the "version" of the slot, like for the `$deposits` storage slot where the "version" value has been update from `1` to `2` to identify a change.

Recommendation: Kiln should consider performing these actions:

- If the storage slot has been removed (deprecated), track the value of the removed slot and if the storage values have been fully cleaned by the migration logic.
- If the storage slot has been added, track the value of the new slot and if the migration logic has migrated any previous value (and from where) or has initialized it with some default value.
- If the storage slot has been "upgraded", track which was the previous value and how the migration logic has migrated the old values to the new one. In addition to that, they should track if the previous storage slot values have been fully cleaned by the migration logic.

In addition to manually documenting the migration changes, Kiln could improve the migration procedure by ensuring that the removed or upgraded storage slot are not used anymore in future upgraded to avoid possible clashes or usage of "dirty" storage slots.

Kiln: Fixed in commit e8941a8a.

Spearbit: Fix verified.

5.2.2 Wrong code comment about the new data structure for `$deposits`

Severity: Informational

Context: vFactory.storage_migration.sol#L165

Description: As we can see in the following code.

```
//  DEPOSIT_OLD              DEPOSIT_NEW
//    owner  ----------------> owner
//    index  X                 dedicatedRecipient = wc == bytes32(0) ? address(0) :
↪  withdrawalAddress(publicKey)
//    withdrawalChannel X      feeRecipient = validator.feeRecipient
//                             threshold = 0
```

According to the documentation `dedicatedRecipient` will be set to `address(0)` in case `wc` is 0 while in practice it is the opposite, in case `wc` is 0 then `dedicatedRecipient = withdrawalAddress(publicKey)`.

Recommendation: Consider changing the comment to:

```
    //  DEPOSIT_OLD              DEPOSIT_NEW
    //    owner  ----------------> owner
    //    index  X                 dedicatedRecipient = wc != bytes32(0) ? address(0) :
    ↪  withdrawalAddress(publicKey)
    //    withdrawalChannel X      feeRecipient = validator.feeRecipient
    //                             threshold = 0
```

Kiln: Fixed in commit a60dd290.

Spearbit: Fix verified.

5.2.3 Old channel state might not be deleted completely

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: As part of the data migration withdrawal channels related to old deposits are being wiped. Let's look at the struct of a withdrawal channel:

```
struct WithdrawalChannel {
    Validator[] validators;
    uint256 lastEdit;
    uint32 limit;
    uint32 funded;
}
```

Let's also consider the following code that iterates over the validators and wipes their data:

```
channel.validators[validatorChannelIndex] = Validator({
    publicKey: PublicKeyUtils.PublicKey({
        A: bytes32(0),
        B: bytes16(0)
    }),
    signature: SignatureUtils.Signature({
        A: bytes32(0),
        B: bytes32(0),
        C: bytes32(0)
    }),
    feeRecipient: address(0)
});
```

As we can see the length of `channel.validators` will not be zeroed and therefore the data will not be completely wiped.

Another potential issue is the fact that in the current script only channels that has `channel.lastEdit > 0` and that are referenced by a deposit are going to be wiped. In case there are channels that do not satisfy these conditions they will stay in storage.

Recommendation: Consider using low level assembly to set the array length of `channel.validators` to 0 as well as delete all the channels.

Kiln: Fixed in commit 7881dc5a.

Spearbit: Fix verified.

5.2.4 `pkr` values might be overwritten in case of duplicated public keys

Severity: Informational

Context: vFactory.storage_migration.sol#L405

Description: `vFactory` v2.2 has a global check that reverts for duplicated public keys:

```
if (pkr[__.publicKeyHash.k()] != 0) {
    revert ValidationKeyAlreadyActivated(__.validationKey);
}
```

However, `vFactory` v2.0 does not check for duplicated keys and therefore might generate duplicates that will be overwritten in the migration script in:

```
pkr[uint256(keccak256(__.pubKey))] = i;
```

Recommendation: Consider reverting the execution of the migration script in case `pkr[uint256(keccak256(__.pubKey))] != 0`.

Kiln: Fixed in commit 0dc41d40.

Spearbit: Fix verified.

5.2.5 `Migration_2_1_base` refactoring

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The `migrate_vFactories` function could be further improved and refactored to waste less gas, be more efficient and increase the readability.

- `totalKeys` does not need to be re-computed every time, it can be computed once during the very first loop that loops the channel's of an operator. At this point, it can be stored inside a local variable that tracks it for every `(operator, vFactory)` tuple. The `totalKeys` represents the total number of validators that have been funded for in a specific `vFactory`. By storing, it won't be needed anymore to re-calculate it every time during the execution of the tests. The `migrationDetails` can be removed given that right now is only used to loop to calculate every time the `totalKeys` value, which won't change during the execution of the `migrate_vFactories` function.
- The integrity checks block of code executed for the new `$deposits` values in Migration.base.t.sol#L1282-L1288 should be refactored. In the current `vFactory` implementation, the `bytes(0)` withdrawal channel is not supported and the `.migrate(...)` found would have reverted already anyway. Replace the referenced block of code with just `expect(d.dedicatedRecipient).toEqual(address(0), "dedicatedRecipient mismatch");`.

Recommendation: Kiln should consider implementing the suggestions listed in the above section.

Kiln: Acknowledged as these points are related mainly to the gas usage and performance of the test suite.

Spearbit: Acknowledged.

5.2.6 Natspec and dev comments inaccuracies in `vFactory.storage_migration`

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description:

- vFactory.storage_migration.sol#L348: The `migrate` function miss the natspec documentation for the `deletionParameters` input parameter.
- vFactory.storage_migration.sol#L390-L391: The comment refers to "recipient salt" which has been removed from `vFactory` v2.2 and has been replaced by `dedicatedRecipient`.

Recommendation: Kiln should add or solve the above suggested natspec changes.

Kiln: Fixed in commit 13681644.

Spearbit: Fix verified.

5.2.7 `vFactory.storage_migration` refactoring and improvements

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description:

1. vFactory.storage_migration.sol#L208-L219: The `Validator` and `WithdrawalChannel` structs do not exist anymore in the `vFactory` v2.1. Consider keeping the same nomenclature and logic applied to the `Deposit` struct and renaming every struct related to `vFactory` v2.0 to follow the pattern `structName_old`. `Validator` would be renamed `Validator_old` and `WithdrawalChannel` to `WithdrawalChannel_old`.
2. vFactory.storage_migration.sol#L394-L397: Consider moving the revert logic in the `if` branch `__.wc == bytes32(0)`, with the whole relative comment `// 4.1.5. `, after the logic that retrieves the `__.wc` value.
3. vFactory.storage_migration.sol#L402: Consider explicitly resetting the new deposit `threshold` to `0` and `dedicatedRecipient` to `address(0)`.
4. vFactory.storage_migration.sol#L434-L437: Move the `wc == bytes32(0)` check and revert just below the `wc` value is initialized in vFactory.storage_migration.sol#L428. This will revert the logic as soon as possible and improve the readability of the code.
5. vFactory.storage_migration.sol#L450: Move the logic that migrates the `$fundedValidators.get()[uint256(wc)] = channel.funded;` value from the `eraseMode == true` branch to the `eraseMode == false` branch. This change is more logical, given that when we are in erase mode, we should only erase and the migration should have been done in the step before.
6. vFactory.storage_migration.sol#L362-L365: The wiping of the `$operator` and `$balances` should be performed when the migration is in `eraseMode == true`.

Recommendation: Kiln should consider applying the above suggested changes.

Kiln: Fixed in commit 45fd0790. Point 1 is acknowledged (we agree it's better from a readability point of view, but doing it this way is simpler and ensures one action per channel).

Spearbit: Point 5 of the recommendations has been acknowledged by Kiln and won't be implemented.

5.2.8 Migration specific storage states will continue to exist in `vFactory` after migration

Severity: Informational

Context: vFactory.storage_migration.sol#L246-L253, vFactory.storage_migration.sol#L417, vFactory.storage_migration.sol#L463

Description: The `vFactory_2_1_storage_migration` contract introduces two new storage state variables `$migration_tracker` & `$erase_mode_tracker`. Once the migration is complete these storage states will have non-zero values as:

- `$migration_tracker = type(uint256).max`.
- `$erase_mode_tracker = true`.

These values will remain persistent in storage of all `vFactory` contracts forever. There are no comment in the `v2.2` implementation of `vFactory` contract which explicitly indicate this behaviour. As the goal of PR 175 was to reset all unused/unnecessary storage states on `vFactory`, leaving these states as non-zero will not be ideal.

Recommendation: Consider resetting the `$migration_tracker` & `$erase_mode_tracker` states at end of migration. The `Initializable.$version` state variable can also be utilised to implement this. Or, leave appropriate comments in `vFactory` contract about this behaviour.

Kiln: Fixed in PR 176.

Spearbit: Fix verified.

5.2.9 Input validation of `vFactory_2_1_storage_migration.migrate` function can be improved

Severity: Informational

Context: vFactory.storage_migration.sol#L298-L310, vFactory.storage_migration.sol#L348

Description: The `vFactory_2_1_storage_migration.migrate` function takes `keyCount` & `deletionParameters` parameters as input but lacks to perform strong validations on them. Currently it allows invalid input parameters to be passed to the `migrate` call. For example, a user can pass:

- `deletionParameters` - duplicate entries in the array.
- `deletionParameters` - non-existent `wc-owner` pair whose `$balances` value is `0`.
- `keyCount` - `0` as `keyCount`.

The `migrate` call executes successfully with these input parameters. However these invalid values do not cause any harm to the `vFactory` contract.

Recommendation: Consider improving the input validation checks of `migrate` function.

Kiln:

- `keyCount` - fixed by PR 176.
- `deletionParameters` - acknowledged, the assumption is that if the provided list contains all the owners (even if there are more and invalid ones) it will work. It can contain 100 invalid entries and it won't fail. If there's a single valid entry missing it will fail.

Spearbit: Partially fixed.

5.2.10 Non-critical issues

Severity: Informational

Context: vFactory.hatcher_fix.sol#L18-L20, vFactory.storage_migration.sol#L246-L253, Cub.sol#L67, TUPProxy.sol#L122

Description: List of issues:

1. `vFactory.storage_migration.sol#L246-L253`: Misleading comment, the states are not copied.
2. `vFactory.hatcher_fix.sol#L18`: Unused `vOracleAggregatorLike` interface can be removed.
3. `Cub.sol#L67`: As per the function's usage it should be marked as `external`.
4. `TUPProxy.sol#L122`: Misleading comment, there is no `_beforeFallback` in `Proxy`.

Kiln: Fixed in commit e9b73136.

Spearbit: Partially fixed (points 3 and 4 have not been addressed).

5.2.11 Potential re-entrancy in `Cub._fix` operation

Severity: Informational

Context: Cub.sol#L97-L106

Description: In the `Cub._fix` function the fix progress is committed on hatcher (`_commit()`) after performing the fixes (`_applyFix()`). This technically can expose the `_fix` operation to re-entrancy risk. As per `_fallback` function, every call to `Cub` always tries to apply any pending fixes. Hence the `_fix` call can be considered as publicly accessible. In case a `fix` call passes the execution control to an untrusted contract then that contract can re-enter the `fallback` function which will again try to apply the pending fix.

Here is an example flow:

```
- Cub.fallback
- Fetch and apply fixes
- Call untrusted contract
    - Cub.fallback
    - Fetch and apply fixes
    - Commit fixes
- Commit fixes
```

Due to this same fixes can be executed multiple times which could be unintended for the `Cub` proxy contract.

Recommendation: Consider performing the `_commit()` operation before the `_applyFix()` operation. Or add a `nonReentrant` modifier.

Kiln: Acknowledged as we can't do anything regarding Cub contracts + the risk is similar to the usual upgrade pattern risk (admin rights can do a lot of damage with invalid upgrades).

Spearbit: Acknowledged.