SPEARBIT

Kiln staking contracts Security Review

Auditors

Saw-mon and Natalie, Lead Security Researcher

Optimum, Lead Security Researcher

Report prepared by: Lucas Goiriz

March 31, 2025

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 Kiln staking contracts 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 2 days in total, Kiln engaged with Spearbit to review the kiln-staking-contracts protocol. In this period of time a total of **9** issues were found.

Summary

Project Name	Kiln
Repository	kiln-staking-contracts
Commit	d3d77a5f
Type of Project	Vaults, Staking
Audit Timeline	Jan 13th to Jan 15th

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	0	0	0
Low Risk	2	2	0
Gas Optimizations	1	0	1
Informational	5	3	2
Total	**9**	**6**	**3**

5 Findings

5.1 High Risk

5.1.1 A sanctioned withdrawer can change the withdraw address of public keys to call `requestValidatorsExit`

Severity: High Risk

Context: StakingContract.sol#L440-L455, StakingContract.sol#L731-L734

Description: If a user is on the sanctioned list of the sanction oracle, ie the call to `_revertIfSanctioned(msg.sender)` reverts and if the withdrawer customisation is enabled, the sanctioned user can call `setWithdrawer` to change the withdraw address associated to the public keys in the question to an address that is not sanctioned then call `requestValidatorsExit` with this new address.

Recommendation: Make sure `setWithdrawer` also reverts for sanctioned users.

Kiln: Fixed in d3a14f20.

Spearbit: Changing the withdrawer of a public key and also enabling withdrawer customisation has been removed from the codebase in d3a14f20 which solves this issue.

5.2 Low Risk

5.2.1 Emitting events are missing

Severity: Low Risk

Context: StakingContract.sol#L212-L214, StakingContract.sol#L376-L378

Description/Recommendation: In this context emitting events are missing.

Kiln: Fixed in commit 730eb947.

Spearbit: Fix verified.

5.2.2 `_account` could potentially cause some calls to `blockAccount` to be reversed

Severity: Low Risk

Context: StakingContract.sol#L746-L755, StakingContract.sol#L832-L834

Description: If withdrawer customization is enabled and a non-empty `_publicKeys` is passed to `blockAccount` by the admin, if the `_account` is not sanctioned, it can frontrun the call to change (in StakingContract.sol#L440) the withdrawer for one of the public keys in the `_publicKeys` so that later in the `_requestExits` flow, the following would revert:

```
address withdrawer = _getWithdrawer(pubKeyRoot);
if (owner != withdrawer) {
    revert Unauthorized();
}
```

`_requestExits` can be either called from:

- `requestValidatorsExit` or...
- `blockAccount`.

Recommendation: Make sure when `_requestExits` is called from the `blockAccount` flow, the above `revert Unauthorized()` is skipped. This can be done by using a `_onlyWithdrawerOrAdmin` variant. It is best to pass the `publicKeyRoot` to:

- `_deployAndWithdraw`.
- `_onlyWithdrawerOrAdmin`.

Kiln: Fixed in commit d3a14f20.

Spearbit: Changing the withdrawer of a public key and also enabling withdrawer customisation has been removed from the codebase in d3a14f20 which solves this issue and would also prevent the code from deriving the public key root multiple times in different flows.

5.3 Gas Optimization

5.3.1 Define a new utility `slice` function for calldata specifically

Severity: Gas Optimization

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

Description/Recommendation: Define a new utility `slice` function for calldata specifically and make sure to add unit and fuzzing tests for both the old and the new implementation:

```
diff --git a/lib/forge-std b/lib/forge-std
index 52715a2..3b1c123 160000
--- a/lib/forge-std
+++ b/lib/forge-std
@@ -1 +1 @@
-Subproject commit 52715a217dc51d0de15877878ab8213f6cbbbab5
+Subproject commit 3b1c123f115b5d9bb371a18cc808c1abfb4369c3
diff --git a/src/contracts/StakingContract.sol b/src/contracts/StakingContract.sol
index 7f1d45d..67ae6f5 100644
--- a/src/contracts/StakingContract.sol
+++ b/src/contracts/StakingContract.sol
@@ -560,8 +560,8 @@ contract StakingContract {
             storage operatorIndexPerValidator =
             ↪  StakingContractStorageLib.getOperatorIndexPerValidator();

         for (uint256 i; i < _keyCount; ) {
-            bytes memory publicKey = BytesLib.slice(_publicKeys, i * PUBLIC_KEY_LENGTH,
↪  PUBLIC_KEY_LENGTH);
-            bytes memory signature = BytesLib.slice(_signatures, i * SIGNATURE_LENGTH,
↪  SIGNATURE_LENGTH);
+            bytes memory publicKey = BytesLib.sliceCallData(_publicKeys, i * PUBLIC_KEY_LENGTH,
↪  PUBLIC_KEY_LENGTH);
+            bytes memory signature = BytesLib.sliceCallData(_signatures, i * SIGNATURE_LENGTH,
↪  SIGNATURE_LENGTH);

             operators.value[_operatorIndex].publicKeys.push(publicKey);
             operators.value[_operatorIndex].signatures.push(signature);
@@ -658,7 +658,7 @@ contract StakingContract {
             revert InvalidPublicKeys();
         }
         for (uint256 i = 0; i < _publicKeys.length; ) {
-            bytes memory publicKey = BytesLib.slice(_publicKeys, i, PUBLIC_KEY_LENGTH);
+            bytes memory publicKey = BytesLib.sliceCallData(_publicKeys, i, PUBLIC_KEY_LENGTH);
             _onlyWithdrawerOrAdmin(publicKey);
             _deployAndWithdraw(publicKey, EXECUTION_LAYER_SALT_PREFIX,
             ↪  StakingContractStorageLib.getELDispatcher());
             unchecked {
@@ -676,7 +676,7 @@ contract StakingContract {
             revert InvalidPublicKeys();
         }
         for (uint256 i = 0; i < _publicKeys.length; ) {
-            bytes memory publicKey = BytesLib.slice(_publicKeys, i, PUBLIC_KEY_LENGTH);
+            bytes memory publicKey = BytesLib.sliceCallData(_publicKeys, i, PUBLIC_KEY_LENGTH);
             _onlyWithdrawerOrAdmin(publicKey);
```

```
            _deployAndWithdraw(publicKey, CONSENSUS_LAYER_SALT_PREFIX,
            ↪  StakingContractStorageLib.getCLDispatcher());
            unchecked {
@@ -693,7 +693,7 @@ contract StakingContract {
            revert InvalidPublicKeys();
        }
        for (uint256 i = 0; i < _publicKeys.length; ) {
-           bytes memory publicKey = BytesLib.slice(_publicKeys, i, PUBLIC_KEY_LENGTH);
+           bytes memory publicKey = BytesLib.sliceCallData(_publicKeys, i, PUBLIC_KEY_LENGTH);
            _onlyWithdrawerOrAdmin(publicKey);
            _deployAndWithdraw(publicKey, EXECUTION_LAYER_SALT_PREFIX,
            ↪  StakingContractStorageLib.getELDispatcher());
            _deployAndWithdraw(publicKey, CONSENSUS_LAYER_SALT_PREFIX,
            ↪  StakingContractStorageLib.getCLDispatcher());
@@ -826,7 +826,7 @@ contract StakingContract {
        }

        for (uint256 i = 0; i < publicKeys.length; ) {
-           bytes memory publicKey = BytesLib.slice(publicKeys, i, PUBLIC_KEY_LENGTH);
+           bytes memory publicKey = BytesLib.sliceCallData(publicKeys, i, PUBLIC_KEY_LENGTH);
            bytes32 pubKeyRoot = _getPubKeyRoot(publicKey);
            address withdrawer = _getWithdrawer(pubKeyRoot);
            if (owner != withdrawer) {
diff --git a/src/contracts/libs/BytesLib.sol b/src/contracts/libs/BytesLib.sol
index 2950055..91a6b63 100644
--- a/src/contracts/libs/BytesLib.sol
+++ b/src/contracts/libs/BytesLib.sol
@@ -141,4 +141,35 @@ library BytesLib {

        return tempBytes;
    }
+
+   function sliceCallData(
+       bytes calldata _bytes,
+       uint256 _start,
+       uint256 _length
+   ) internal pure returns (bytes memory tempBytes) {
+       require(_length + 31 >= _length, "slice_overflow");
+       require(_bytes.length >= _start + _length, "slice_outOfBounds");
+
+       assembly {
+           // Get a location of some free memory and store it in tempBytes as
+           // Solidity does for memory variables.
+           tempBytes := mload(0x40)
+
+           let cs := add(_bytes.offset, _start)
+           let cc := add(cs, _length)
+           let mc := add(tempBytes, add(0x20, _length))
+
+           //update free-memory pointer
+           //allocating the array padded to 32 bytes like the compiler does now
+           mstore(0x40, and(add(mc, 31), not(31)))
+
+           for {} gt(cc, cs) {} {
+               cc := sub(cc, 0x20)
+               mc := sub(mc, 0x20)
+               calldatacopy(mc, cc, 0x20)
+           }
+
+           mstore(tempBytes, _length)
+       }
+   }
}
```

forge s --diff:

```
testWithdrawAllFees() (gas: -224 (-0.035%))
testWithdrawAllFees() (gas: -224 (-0.037%))
testWithdrawCLFeesAlreadyDeployed() (gas: -224 (-0.042%))
testWithdrawCLFeesExitedValidator_UserTriesToStealFee() (gas: -224 (-0.043%))
testWithdrawCLFeesAlreadyDeployed() (gas: -224 (-0.044%))
testWithdrawCLFeesEditedOperatorFee() (gas: -224 (-0.045%))
testWithdrawCLFeesExitedValidator_RewardsAfterRequest() (gas: -224 (-0.047%))
testWithdrawCLFeesExitedValidator() (gas: -224 (-0.047%))
testWithdrawCLFeesEditedOperatorFee() (gas: -224 (-0.048%))
testBatchWithdrawAllFees_asAdmin() (gas: -462 (-0.048%))
testBatchWithdrawAllFees() (gas: -462 (-0.048%))
testWithdrawCLFeesExitedValidator() (gas: -224 (-0.049%))
testWithdrawCLFeesSlashedValidatorWithRewards() (gas: -224 (-0.050%))
testBatchWithdrawAllFees_WrongWithdrawer() (gas: -461 (-0.068%))
testBatchWithdrawCLFees_asAdmin() (gas: -462 (-0.069%))
testBatchWithdrawCLFees() (gas: -462 (-0.069%))
testBatchWithdrawELFees_asAdmin() (gas: -462 (-0.071%))
testBatchWithdrawELFees() (gas: -462 (-0.071%))
testBatchWithdrawAllFees_10() (gas: -3219 (-0.080%))
testBatchWithdrawCLFees_WrongSecondWithdrawer() (gas: -461 (-0.088%))
testBatchWithdrawELFees_WrongWithdrawerSecondKey() (gas: -461 (-0.090%))
test_block_UserDepositOneValidator_NotSanctioned() (gas: -224 (-0.103%))
test_block_UserDepositOneValidator_NotSanctioned_WrongPublicKey() (gas: -224 (-0.103%))
testRequestValidatorsExits_OneValidator() (gas: -224 (-0.118%))
testBatchWithdrawCLFees_10() (gas: -3169 (-0.122%))
testBatchWithdrawELFees_10() (gas: -3169 (-0.126%))
testRequestValidatorsExits_WrongWithdrawer() (gas: -224 (-0.137%))
testRequestValidatorsExits_TwoValidators() (gas: -461 (-0.149%))
testRequestValidatorsExits_WrongSecondWithdrawer() (gas: -461 (-0.162%))
testDistribution(uint8) (gas: -4503 (-0.226%))
testAddOperatorLimitReached(uint128) (gas: -7649 (-0.319%))
testLoopedDeposit() (gas: -12141000 (-0.361%))
testAddValidatorsOperatorOne() (gas: -7649 (-0.393%))
testAddValidatorsDeactivatedOperatorOne() (gas: -7649 (-0.395%))
testAddValidatorsOperatorOneDuplicateKeys() (gas: -8345 (-0.404%))
testSetOperatorLimitDeactivated(uint256,uint8) (gas: 156457 (2.697%))
testSetOperatorLimit(uint256,uint8) (gas: 156457 (2.701%))
testSetOperatorLimit_snapshotRevert(uint256,uint8) (gas: -16840934 (-37.234%))
Overall gas change: -28722809 (-0.824%)
```

Kiln: We acknowledge it and chose not to apply those changes to limit changes.

Spearbit: Acknowledged.

5.4 Informational

5.4.1 `slice_overflow` error in `BytesLib.slice` cannot be reached

Severity: Informational

Context: BytesLib.sol#L85-L86

Description: In `solc 0.8.x` arithmetic operations revert on over/underflows and so the following expression is either `true` or would revert:

```
_length + 31 >= _length
```

Thus, the error message `slice_overflow` would not ever be thrown below:

```
require(_length + 31 >= _length, "slice_overflow");
```

Proof of Concept:

```
// file: src/test/units/BytesLib.t.sol
pragma solidity >=0.8.10;

import "forge-std/Test.sol";
import "forge-std/StdError.sol";
import "../../contracts/libs/BytesLib.sol";

contract BytesLibTest is Test {

    function testSliceOverflow(
        bytes memory _bytes,
        uint256 _start,
        uint256 _length
    ) public {
        vm.assume(_length > type(uint256).max - 31);

        //vm.expectRevert(stdError.arithmeticError); // <--- currently this is triggered
        vm.expectRevert("slice_overflow");
        bytes memory res = BytesLib.slice(_bytes, _start, _length);
        console.logBytes(res);
    }
}
```

Recommendation: Instead one can use the following check:

```
require(_length < type(uint256).max - 30, "slice_overflow");
```

or since the next line check limits the upper bound for `_length` by `_bytes.length` one can add an upper bound restriction for the `_bytes.length` instead of the line in question.

Kiln: We acknowledge it and as the behavior is still the expected one even if the exact error isn't we chose not to apply the suggested changes.

Spearbit: Acknowledged.

5.4.2 Unused code

Severity: Informational

Context: StakingContract.sol#L56-L62, StakingContract.sol#L970-L975

Description/Recommendation: The code snippets in this context are not used and can be removed.

Kiln: Fixed in commit d3a14f20.

Spearbit: Fix verified.

5.4.3 Length check in `addValidators` can be simplified

Severity: Informational

Context: StakingContract.sol#L550-L556

Description/Recommendation: These length checks can be simplified to:

```
if (_publicKeys.length != PUBLIC_KEY_LENGTH * _keyCount) {
    revert InvalidPublicKeys();
}

if (_signatures.length != SIGNATURE_LENGTH * _keyCount) {
    revert InvalidSignatures();
}
```

Kiln: Fixed in commit a268ce58.

Spearbit: Fixed.

5.4.4 To-be-sanctioned withdrawers can always frontrun the sanctioning call to the saction oracles

Severity: Informational

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

Description: Sanction oracles such as chainalysis have endpoints like `addToSanctionsList` where their admins need to call to add an address to the sanction list. If the to-be-sanctioned withdrawer observes this transaction in a public mempool they can frontrun it with a call to `setWithdrawer` (if the withdrawer customisation is enabled) to check their address to a one not on the to be updated sanction list.

Recommendation: This is something to note as different types of interactions would be needed between the sanction oracle and the staking contract. At list the above needs to be monitored and if such behaviours are noticed the staking contract should disable its withdrawer customisation while the sanction oracle adds the newly frontrunned changed address to the list then the staking contract can toggle its withdrawer customisation on plus the modification to the `setWithdrawer` that would revert for sanctioned users.

Kiln: Fixed in commit d3a14f20.

Spearbit: Changing the withdrawer of a public key and also enabling withdrawer customisation has been removed from the codebase in d3a14f20 which solves this issue.

5.4.5 Other addresses in the staking contracts are not checked against the blocked and sanctioned lists

Severity: Informational

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

Description/Recommendation: Other addresses in the staking contracts are not checked against the blocked and sanctioned lists. These addresses include:

- Pending and new admins.
- Operators.
- Operator fee recipients.
- Treasury.

Kiln: We acknowledge this issue.

Spearbit: Acknowledged.