CANTINA

Ethereum Pectra: Lodestar

Competition

July 21, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

A competition provides a broad evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While competitions endeavor to identify and disclose all potential security issues, they cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities, therefore, any changes made to the code would require an additional security review. Please be advised that competitions are not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
High	*Must* fix as soon as possible (if already deployed) and can be triggered by any user without significant constraints, generating outsized returns to the exploiter. For example: loss of user funds (significant amount of funds being stolen or lost) or breaking core functionality (failure in fundamental protocol operations).
Medium	Global losses <10% or losses to only a subset of users, requiring significant constraints (capital, planning, other users...) to be exploited. For example: temporary disruption or denial of service (DoS), minor fund loss or exposure or breaking non-core functionality
Low	Losses will be annoying but easily recoverable, requiring unusual scenarios or admin actions to be exploited.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above matrix. High severity findings represent the most critical issues that must be addressed immediately, as they either have high impact and high likelihood of occurrence, or medium impact with high likelihood.

Medium severity findings represent issues that, while not immediately critical, still pose significant risks and should be addressed promptly. These typically involve scenarios with medium impact and medium likelihood, or high impact with low likelihood.

Low severity findings represent issues that, while not posing immediate threats, could potentially cause problems in specific scenarios. These typically involve medium impact with low likelihood, or low impact with medium likelihood.

Lastly, some findings might represent improvements that don't directly impact security but could enhance the codebase's quality, readability, or efficiency (Gas and Informational findings).

2 Security Review Summary

Ethereum is a worldwide system, an open-source platform to write computer code that stores and automates digital databases using smart contracts, without relying upon a central intermediary, solving trust with cryptographic techniques.

From Feb 21st to Mar 27th Cantina hosted a competition based on the Ethereum Pectra upgrade. The present report focuses in the lodestar implementation. The participants identified **1** issue in the following risk category:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 1
- Gas Optimizations: 0
- Informational: 0

3 Findings

3.1 Low Risk

3.1.1 EIP7251 - Not checking if the consolidation request originates from execution withdrawal credentials may result in a chain split

Submitted by franfran

Severity: Low Risk

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

Summary: As EIP7251 introduces consolidation requests, the new `process_consolidation_request` function should make sure that the source validator has execution withdrawal credentials (prefix `0x01` or `0x02`). Since it's not checked, lodestar will accept consolidation requests from withdrawal credentials with the `BLS_WITHDRAWAL_PREFIX` while other clients should reject it, effectively creating a chain split.

Finding Description: Here is the consensus specification of the new `process_consolidation_request` function which processes consolidation requests. There is a check to reject invalid consolidation requests, if the source validator doesn't have the right prefix, a legacy BLS withdrawal prefix for instance.

```
# Verify withdrawal credentials
has_correct_credential = has_execution_withdrawal_credential(validator)
is_correct_source_address = (
    validator.withdrawal_credentials[12:] == withdrawal_request.source_address
)
if not (has_correct_credential and is_correct_source_address):
    return
```

And the resulting check to `has_execution_withdrawal_credential` in the specs is not done in lodestar:

```
const sourceValidator = state.validators.get(sourceIndex);
const targetValidator = state.validators.getReadonly(targetIndex);
const sourceWithdrawalAddress = sourceValidator.withdrawalCredentials.subarray(12);
const currentEpoch = state.epochCtx.epoch;


// Verify that target has compounding withdrawal credentials
if (!hasCompoundingWithdrawalCredential(targetValidator.withdrawalCredentials)) {
  return;
}


if (Buffer.compare(sourceWithdrawalAddress, sourceAddress) !== 0) {
  return;
}
```

Resulting in consolidation requests that will pass for Lodestar clients, while other clients should reject them, which may cause a chain split for Lodestar users.

Impact Explanation: High - Permanent hard fork for Lodestar users.

Likelihood Explanation: High - This attack only requires to send a a consolidation request from an invalid source validator.

Proof of Concept:

```
diff --git a/packages/state-transition/test/unit/block/processConsolidationRequest.test.ts
↪  b/packages/state-transition/test/unit/block/processConsolidationRequest.test.ts
index e69de29bb2..c7569d0210 100644
--- a/packages/state-transition/test/unit/block/processConsolidationRequest.test.ts
+++ b/packages/state-transition/test/unit/block/processConsolidationRequest.test.ts
@@ -0,0 +1,61 @@
+import {processConsolidationRequest} from "../../../src/block";
+import {describe, expect, it} from "vitest";
+import {ssz} from "@lodestar/types";
+import {
+  BLS_WITHDRAWAL_PREFIX, COMPOUNDING_WITHDRAWAL_PREFIX,
+  FAR_FUTURE_EPOCH,
+  MAX_EFFECTIVE_BALANCE,
+  SLOTS_PER_EPOCH,
+  SYNC_COMMITTEE_SIZE
+} from "@lodestar/params";
+import {generateValidators} from "../../utils/validator.js";
+import {generateCachedElectraState} from "../../../../beacon-node/test/utils/state.js";
+import bls from "@chainsafe/blst";
+import {digest} from "@chainsafe/as-sha256";
+
+describe.only("processConsolidationRequest", () => {
+  it.only("rejects BLS withdrawal credentials", () => {
+    const electraForkEpoch = 400000;
+    const currentEpoch = electraForkEpoch + 10;
+    const currentSlot = SLOTS_PER_EPOCH * currentEpoch;
+
+    const activationEpoch = electraForkEpoch - 10000;
+    const exitEpoch = FAR_FUTURE_EPOCH;
+
+    const validatorOpts = {
+      activationEpoch,
+      activation: activationEpoch,
+      effectiveBalance: MAX_EFFECTIVE_BALANCE,
+      withdrawableEpoch: FAR_FUTURE_EPOCH,
+      exitEpoch,
+      exit: exitEpoch,
+    };
+    const validators = generateValidators(SYNC_COMMITTEE_SIZE, validatorOpts);
+    for (let i = 0; i < SYNC_COMMITTEE_SIZE; i++) {
+      const buffer = Buffer.alloc(32, 0);
+      buffer.writeInt16BE(i + 1, 30); // Offset to ensure the SK is less than the order
+      const sk = bls.SecretKey.fromBytes(buffer);
+      validators[i].pubkey = sk.toPublicKey().toBytes();
+    }
+
+    let [sourceValidator, targetValidator] = [validators[0], validators[1]];
+    sourceValidator.withdrawalCredentials = digest(sourceValidator.pubkey);
+    sourceValidator.withdrawalCredentials[0] = BLS_WITHDRAWAL_PREFIX;
+    targetValidator.withdrawalCredentials = digest(targetValidator.pubkey);
+    targetValidator.withdrawalCredentials[0] = COMPOUNDING_WITHDRAWAL_PREFIX;
+
+    let state = generateCachedElectraState({slot: currentSlot + 1, validators}, electraForkEpoch);
+    state.epochCtx.totalActiveBalanceIncrements = 123456789;
+    const request = ssz.electra.ConsolidationRequest.defaultValue();
+
+    request.sourcePubkey = sourceValidator.pubkey;
+    request.targetPubkey = targetValidator.pubkey;
+    request.sourceAddress = digest(sourceValidator.pubkey).slice(12);
+
+    expect(state.pendingConsolidations.length).eq(0);
+
+    processConsolidationRequest(state, request);
+
+    expect(state.pendingConsolidations.length).eq(1);
+  })
+});
```

Run the test with:

```
lerna run --scope @lodestar/state-transition test:unit
```

And observe it pass.

```
# ...
@lodestar/state-transition:   test/unit/block/processConsolidationRequest.test.ts >
↪  processConsolidationRequest > rejects BLS withdrawal credentials 1169ms
#...
```

Recommendation: Make sure to check the prefix of the source validator as in the consensus specs:

```
diff --git a/packages/state-transition/src/block/processConsolidationRequest.ts
↪  b/packages/state-transition/src/block/processConsolidationRequest.ts
index 8d3cfdc062..e040b3e3d5 100644
--- a/packages/state-transition/src/block/processConsolidationRequest.ts
+++ b/packages/state-transition/src/block/processConsolidationRequest.ts
@@ -3,7 +3,12 @@ import {electra, ssz} from "@lodestar/types";

 import {CachedBeaconStateElectra} from "../types.js";
 import {hasEth1WithdrawalCredential} from "../util/capella.js";
-import {hasCompoundingWithdrawalCredential, isPubkeyKnown, switchToCompoundingValidator} from
↪  "../util/electra.js";
+import {
+  hasCompoundingWithdrawalCredential,
+  hasExecutionWithdrawalCredential,
+  isPubkeyKnown,
+  switchToCompoundingValidator
+} from "../util/electra.js";
 import {computeConsolidationEpochAndUpdateChurn} from "../util/epoch.js";
 import {getConsolidationChurnLimit, getPendingBalanceToWithdraw, isActiveValidator} from
 ↪  "../util/validator.js";

@@ -54,7 +59,9 @@ export function processConsolidationRequest(
     return;
   }

-  if (Buffer.compare(sourceWithdrawalAddress, sourceAddress) !== 0) {
+  const hasCorrectCredential = hasExecutionWithdrawalCredential(sourceValidator.withdrawalCredentials);
+  const isCorrectSourceAddress = Buffer.compare(sourceWithdrawalAddress, sourceAddress) === 0;
+  if (!(hasCorrectCredential && isCorrectSourceAddress)) {
     return;
   }
```