CANTINA

Ethereum Pectra: Lighthouse

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 lighthouse implementation. The participants identified a total of **1** issues in the following risk categories:

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

3 Findings

3.1 High Risk

3.1.1 `Lighthouse` might not accurately update effective balances

Submitted by alexfilippov314

Severity: High Risk

Context: single_pass.rs#L993

Description: The single-pass epoch processing implementation in `Lighthouse` may cause the effective balance of some validators to be updated multiple times during epoch processing. For example, the effective balance could be updated once after processing pending deposits (single_pass.rs#L320-L330) and a second time after processing consolidation requests (single_pass.rs#L1101-L1111). The spec states that effective balance updates should be processed only once and after pending consolidations.

```
def process_epoch(state: BeaconState) -> None:
    process_justification_and_finalization(state)
    process_inactivity_updates(state)
    process_rewards_and_penalties(state)
    process_registry_updates(state)  # [Modified in Electra:EIP7251]
    process_slashings(state)  # [Modified in Electra:EIP7251]
    process_eth1_data_reset(state)
    process_pending_deposits(state)  # [New in Electra:EIP7251]
    process_pending_consolidations(state)  # [New in Electra:EIP7251]
    process_effective_balance_updates(state)  # [Modified in Electra:EIP7251]
    process_slashings_reset(state)
    process_randao_mixes_reset(state)
    process_historical_summaries_update(state)
    process_participation_flag_updates(state)
    process_sync_committee_updates(state)
```

The issue arises because updating the effective balance twice does not necessarily produce the same result as a single update due to hysteresis:

```
def process_effective_balance_updates(state: BeaconState) -> None:
    # Update effective balances with hysteresis
    for index, validator in enumerate(state.validators):
        balance = state.balances[index]
        HYSTERESIS_INCREMENT = uint64(EFFECTIVE_BALANCE_INCREMENT // HYSTERESIS_QUOTIENT)
        DOWNWARD_THRESHOLD = HYSTERESIS_INCREMENT * HYSTERESIS_DOWNWARD_MULTIPLIER
        UPWARD_THRESHOLD = HYSTERESIS_INCREMENT * HYSTERESIS_UPWARD_MULTIPLIER
        # [Modified in Electra:EIP7251]
        max_effective_balance = get_max_effective_balance(validator)

        if (
            balance + DOWNWARD_THRESHOLD < validator.effective_balance
            or validator.effective_balance + UPWARD_THRESHOLD < balance
        ):
            validator.effective_balance = min(balance - balance % EFFECTIVE_BALANCE_INCREMENT,
            ↪  max_effective_balance)
```

For example, if the initial balance is 64 ETH, and we process a balance decrease of 32 ETH followed by increase of 31.9 ETH, the outcome depends on how many times the effective balance is updated:

- Single Update (Correct Behavior): The net decrease of 0.1 ETH does not meet the `DOWNWARD_THRESHOLD`, so the effective balance remains 64 ETH.
- Multiple Updates (Incorrect Behavior): If the effective balance is updated twice, both balance changes trigger an update. The second update sets the effective balance to 63 ETH because a balance of 63.9 ETH does not justify an effective balance of 64 ETH.

Consider a scenario where this issue could potentially be triggered:

1. There are two validators, `A` and `B`, where `A` has a balance of 31.9 ETH (validator can still be active with such balance) and `B` has a balance of 64 ETH.
2. `B` submits a partial withdrawal request for 32 ETH.
3. Around the same time, `A` submits a consolidation request to transfer its balance to `B` (`A` $\rightarrow$ `B`).

4. This situation is possible because the `process_consolidation_request` function does not check whether the target validator has any pending withdrawals.
5. It is also possible for both the partial withdrawal and the consolidation processing to occur within the same epoch. This means that before epoch processing, the balance of validator `B` is 32 ETH (since partial withdrawal was already processed), while the effective balance remains 64 ETH.
6. In such situation, according to the spec, `B`'s effective balance after epoch processing should remain 64 ETH because the balance decrease of 0.1 ETH does not meet the threshold required to trigger an effective balance update due to hysteresis.
7. However, in Lighthouse, the outcome differs due to multiple effective balance updates. First, Lighthouse reduces `B`'s effective balance to 32 ETH before processing the consolidation request. Then, after consolidation, it updates `B`'s effective balance to 63 ETH because the new balance of 63.9 ETH does not justify an effective balance of 64 ETH.

Essentially, this means that in such a scenario, Lighthouse updates the effective balance of validator `B` to 63 ETH, even though the correct outcome according to the spec is 64 ETH. This discrepancy will lead to a chain split if the situation occurs.

Recommendation: Ensuring that multiple effective balance updates produce the correct result appears fragile. Consider refactoring the code to guarantee that each validator's effective balance is updated only once.