Ethereum Foundation

Pectra System Contracts Bytecode

Security Assessment Report

Version: 1.0

January, 2025

Contents

Introduction **2**
- Disclaimer . . . 2
- Document Structure . . . 2
- Overview . . . 2

Security Assessment Summary **3**
- Scope . . . 3
- Approach . . . 3
- Coverage Limitations . . . 3
- Findings Summary . . . 3

Detailed Findings **4**

Summary of Findings **5**
- Many Inserts Can Take Place In One Block At Low Gas Fees . . . 6
- ETH Can Be Sent To The Contract . . . 8
- Fee Calculation Can Overflow . . . 9
- Inconsistencies With Specification In EIPs . . . 10
- Overflows When Block Number is Zero . . . 11
- Miscellaneous General Comments . . . 12

A Test Suite **14**

B Vulnerability Severity Classification **15**

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Ethereum Foundation smart contracts. The review focused solely on the security aspects of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract in scope. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Ethereum Foundation smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Ethereum Foundation smart contracts in scope.

Overview

The assessment focused on the following:

1. **EIP-2935**: This contract contains a ring buffer of historical block hashes and two code paths, `get` and `set`, which read from the ring buffer or add to it.
2. **EIP-7002**: This contract contains a queue of staking withdrawal requests and three code paths that depend on the caller and input. A normal caller can add a withdrawal request or retrieve the current fee. The system can call to process requests. Excess withdrawal requests beyond what the consensus layer can process are queued and a fee is calculated to prevent spam requests.
3. **EIP-7251**: This contract contains a queue of requests to consolidate validators in response to a consensus layer parameter change. It contains three code paths depending on the caller and input. A normal caller can add a consolidation request or retrieve the current fee. The system can call to process requests. Excess consolidation requests beyond what the consensus layer can process are queued and a fee is calculated to prevent spam requests, similar to **EIP-7002**.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the `lightclient/sys-asm` repository.

The scope of this time-boxed review was strictly limited to files at commit 1d67916 relating to EIP-2935, EIP-7002 and EIP-7251.

Approach

The manual review focused on verifying the security and correctness of the deployed smart contracts. This involved reverse engineering the bytecode to correlate its behavior against the intended functionality described in the contract's documentation.

Static analysis was performed to inspect the control flow graph, identify critical opcodes and verify how the contracts handle input validation, error propagation and gas consumption.

Dynamic testing simulated various edge cases to evaluate runtime behaviour, including event emissions and storage updates.

Coverage Limitations

Due to the time-boxed nature of this review, all documented vulnerabilities reflect best effort within the allotted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 6 issues during this assessment. Categorised by their severity:

- Medium: 1 issue.
- Low: 1 issue.
- Informational: 4 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Ethereum Foundation smart contracts in scope. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
PEC-01	Many Inserts Can Take Place In One Block At Low Gas Fees	Medium	Open
PEC-02	ETH Can Be Sent To The Contract	Low	Open
PEC-03	Fee Calculation Can Overflow	Informational	Open
PEC-04	Inconsistencies With Specification In EIPs	Informational	Closed
PEC-05	Overflows When Block Number is Zero	Informational	Open
PEC-06	Miscellaneous General Comments	Informational	Open

PEC-01	Many Inserts Can Take Place In One Block At Low Gas Fees		
Asset	`consolidations/main.eas`, `withdrawals/main.eas`		
Status	**Open**		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

For both `EIP-7002` and `EIP-7251`, it is possible to queue a very large number of new requests in a single block. The only validity checking in the contract is that the fee is paid and that the data is correctly formatted. It is therefore easy to submit a large number of requests without needing to operate a validator.

This circumvents the intended restriction of the fee mechanism because the fees only update between blocks. Within a single block, the fee will not change, regardless of the number of new submissions that the contract receives.

The impact of this form of attack is limited as the attacker would still need to pay large quantities of gas. In testing, a single request costs roughly 75k gas. Meaning that it is possible to submit around 400 new requests within a gas limit of 30 million.

However, an attacker can perform some optimisations to lower this gast cost. Roughly 88% of the gas cost of a request comes from 3 `SSTORE` instructions. These are used to store the input data from the user and the address of the user in storage. One way to lower the cost of these `SSTORE` operations is by not writing new data to the slot, but instead writing the value that is already in the slot. This is easy for the first two `SSTORE` operations, an attacker simply changes the input data to the data that is already in the slot. When doing so, the cost of a request drops to roughly 35k gas. The third `SSTORE` writes the address of the caller. In order to optimise this, the attacker would have to have their address already written to this slot. This can be done by repeating this attack multiple times, allowing for more requests each iteration as the attacker's address is written to more slots each time and the average cost per request drops. With this technique the cost of a request is roughly 15k gas. Allowing a theoretic total of 2000 requests in a block.

Consider the following possible impacts of this form of attack:

1. **Fee Avoidance –** A large protocol with many validators would be wise to batch up their consolidation and withdrawal requests in this manner to give all their users the benefits of the lowest available fees. This could lead to many entries being submitted at lower than intended fees.
2. **Storage Bloat –** The development team stated that a strong motivation for the fees was to limit the use of storage space on the execution layer. If the beneficial pattern for users is to submit very large numbers of requests in a single block, then this would lead to heavier, not lighter, usage of the contract's storage space.
3. **Denial-of-Service And Unreasonable Fees –** The potential effect of a large number of submissions in a single block is to inflate the fee and create a backlog of requests that have to be processed. For example, if an attacker sends 2000 `EIP-7002` requests in a single block, it would take 25 minutes to process these requests, delaying all the subsequent requests in the meantime. Additionally, the fee would be inflated to higher than 1 ETH for 130 minutes. `EIP-7251` requests are slightly more gas-intensive but are processed much slower. Given 1650 requests for `EIP-7251` are made in a single block, there would be a delay of 330 minutes and an inflated fee > 1 ETH for 190 minutes. These types of attacks may also occur unintentionally on a smaller scale when a large protocol sends many requests in a single block.
4. **Fee Griefing** Consider a submission contract similar to those in the EIPs:

```
uint256 fee = uint256(bytes32(feeData));

// Add the request.
bytes memory callData = abi.encodePacked(pubkey, amount);
(bool writeOK,) = WithdrawalsContract.call{value: fee}(callData);
```

If a submission to a contract such as this were to appear in the mempool, it could be frontrun by transactions in a previous block that pump the fee to potentially thousands of ETH, which this contract would then pay. Of course, a contract holding thousands of ETH should check the fee before sending, as should any contract, but it remains a potential attack vector.

Recommendations

Consider limiting the number of submissions per block for these contracts.

PEC-02	ETH Can Be Sent To The Contract		
Asset	`execution_hash/main.eas`		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Medium

Description

The implementation of `EIP-2935`'s predeploy contract has no code paths that require payment of a fee. However, it is possible to make a call with an attached `value` to the contract when requesting a block hash.

This presents an issue because the contract is marked as `payable` (in Solidity terms) but does not include a mechanism to recover or manage funds that are accidentally sent to it.

Recommendations

Consider adding a check at the beginning of the contract which reverts if `CALLVALUE` is greater than zero to prevent ETH from being sent to the contract.

PEC-03	Fee Calculation Can Overflow
Asset	`common/fake_expo.eas`
Status	**Open**
Rating	Informational

Description

Given a `factor` of 1 and a `denominator` of 17, like in these contracts, the fake exponentiation algorithm begins to overflow at a `numerator` of 2893.

This is because, at the `accum*numer` step of the logic, the numerator reaches over `10^74` and this, multiplied by 2893, is greater than `type(uint256).max`.

As such, for an `excess` beyond this number, the fee will overflow and fluctuate, although no low values were observed for a numerator below 10,000 (and possibly well beyond).

Given the numbers in PEC-01, an `excess` of 2893 may be within reach given some extra optimisations or a higher block gas limit. However, seeing as there is no direct impact, the testing team rates this issue as informational.

Recommendations

Ensure the above comments are understood and consider changes if desired.

PEC-04	Inconsistencies With Specification In EIPs
Asset	`consolidations/main.eas`, `withdrawals/main.eas`
Status	**Closed:** See Resolution
Rating	Informational

Description

There are several discrepancies between the EIP specifications and the corresponding bytecode:

- For the read path, `EIP-7002` specifies that *"if the input length is zero, return the current excess withdrawal requests count"*. However, the bytecode returns the current fee (and the comments in the `eas` file match this).
- Similarly, `EIP-7251` specifies that, *"if the input length is zero, return the current excess consolidation requests count"* and the comment on line [**78**] of `main.eas` mentions, *"This is the read path, where we return the current excess"*. However, it returns the current request fee, not the value stored in `SLOT_EXCESS`.
- The buffer length in `EIP-2935` is noted as 8192, however in the bytecode it is 8191.
- The bytecode in `EIP-2935` does not match the bytecode assessed at the given commit.
- `EIP-2935` mentions that *"For any output outside the range of `[block.number-HISTORY_SERVE_WINDOW, block.number-1]` return 0"*. However, instead of returning 0, the execution is reverted in this case.

Recommendations

Review these inconsistencies and either modify the bytecode or otherwise note the updates to the specification.

Resolution

The development team confirmed that the functionality implemented in the bytecode, not the one documented in the EIPs, was desired.

PEC-05	Overflows When Block Number is Zero
Asset	`execution_hash/main.eas`
Status	**Open**
Rating	Informational

Description

When `block.number` is zero, several overflows occur:

- On the `set` path, the storage slot calculation overflows on line [**88**]. However, according to the `EIP-2935` spec, the system contract is not meant to be executed at genesis, so there is no direct impact.
- On the `get` path, the checks on line [**45-52**] and line [**56-61**] may be bypassed. For example, a request for the out-of-range block $2^{256} - 50$ will execute successfully and will not revert as expected. However, the returned blockhash will still be zero as no data is written to the slots yet. As such this issue, has no significant impact and has been assigned an informational severity.

Recommendations

Ensure the above comments are understood and consider changes if desired.

PEC-06	Miscellaneous General Comments
Asset	`execution_hash/main.eas`, `consolidations/main.eas`, `withdrawals/main.eas`
Status	Open
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications.

Note, for brevity, from here on `execution_hash` is used to refer to the file `execution_hash/main.eas`, etc.

1. **Comment Issues**
   - `execution_hash` line [**40**] comment says that, *"execution jumps to continue otherwise revert"*, whereas the actual logic is the inverse.
   - `execution_hash` line [**55**] comment stops in mid sentence.
   - `withdrawals` line [**298**] `[mask, pk2_am, offset, ..]` should be `[mask, pk2_am, pk2_am, offset, ..]`.
   - `withdrawals` line [**349**] the value stored in `QUEUE_HEAD` is referenced as `head_slot` whereas in the rest of the file is it referred to as `head_idx_slot`.
   - `withdrawals` line [**353**] the value stored in `QUEUE_TAIL` is referenced as `tail_slot` whereas in the rest of the file is it referred to as `tail_idx_slot`.
   - `withdrawals` line [**363-370**] there is an excess `count` at the bottom of the stack.
   - `withdrawals` line [**381**] is missing the value `count` at the bottom of the stack (in addition to the previous item).
   - `consolidations` line [**55**], line [**105**], line [**109**], line [**118**] have no stack comment.
   - `consolidations` line [**214**] comment is not quite accurate. It should be words to the effect of, *"determine if count is less than the max requests and use it if so."*
   - `consolidations` line [**302**] stack comment should start `[2,slotbase`.
2. **Typos**
   - `execution_hash` line [**55**] *"verfied"* should be *"verified"*.
   - `consolidations` line [**6**] *"stoarge"* should be *"storage"*.
   - `consolidations` line [**299**] *"triggerred"* should be *"triggered"*.
3. **Unclear Naming** – In `withdrawals` consider renaming `QUEUE_HEAD` and `QUEUE_TAIL` to `SLOT_QUEUE_HEAD` and `SLOT_QUEUE_TAIL` respectively to clearly indicate they are slot keys, analogous to `SLOT_EXCESS` and `SLOT_COUNT`.
4. **Two Values Are Named "count" In The Comments** – In both `withdrawals` from line [**375**] and `consolidations` from line [**377**], the request count (from `WITHDRAWAL_REQUEST_COUNT_STORAGE_SLOT` or `CONSOLIDATION_REQUEST_COUNT_STORAGE_SLOT`) is given the same variable name `count` as the iteration `count` from this call. This makes the code harder to read and also could potentially lead to coding errors.
5. **Hardcoded Value** – In `withdrawals` on line [**237**] the hardcoded value `3` is used instead of `SLOTS_PER_ITEM`.

6. **Unnecessary Stack Cleaning –** `consolidations` line [**348-9**] appear to be superfluous. The excess `new_head_idx` could simply be left at the bottom of the stack.
7. **Unnecessary Steps –** In `withdrawals` there is a slight inefficiency around line [**300-315**]. At line [**303**], it would be possible to convert `pk2_am` to `am` before duplicating `offset` to the top of the stack. Instead, `offset` is duplicated to the top of the stack, modified, swapped with `pk2_am` and latter swapped back.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `forge` framework was used to perform these tests and the output is given below.

```
Ran 3 tests for test/ExecutionHashSigP.t.sol:ContractSigPTest
[PASS] testEmptyUpdate() (gas: 22527)
[PASS] testUpdateRead() (gas: 31393)
[PASS] testValueRead() (gas: 14092)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 28.74ms (246.25µs CPU time)

Ran 9 tests for test/ConsolidationSigP.t.sol:ConsolidationSigPTest
[PASS] testConsolidation() (gas: 200291)
[PASS] testFee() (gas: 25691231)
[PASS] testFeeAttack() (gas: 238909969)
[PASS] testFeeDecreases() (gas: 182061589)
[PASS] testFeeIncreases() (gas: 371578918)
[PASS] testInhibitorInhibits() (gas: 4993)
[PASS] testInhibitorReset() (gas: 31627)
[PASS] testInvalidRequest() (gas: 27781)
[PASS] testQueueReset() (gas: 1598279)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 635.33ms (1.37s CPU time)

Ran 9 tests for test/WithdrawalsSigP.t.sol:WithdrawalsSigPTest
[PASS] testFee() (gas: 382683453)
[PASS] testFeeAttack() (gas: 130324770)
[PASS] testFeeDecreases() (gas: 117579088)
[PASS] testFeeIncreases() (gas: 198434063)
[PASS] testInhibitorInhibits() (gas: 4949)
[PASS] testInhibitorReset() (gas: 21180)
[PASS] testInvalidWithdrawal() (gas: 27677)
[PASS] testQueueReset() (gas: 5604620)
[PASS] testWithdrawal() (gas: 156966)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 774.65ms (1.67s CPU time)

Ran 1 test for test/FakeExpoSigP.t.sol:FakeExpoSigPTest
[PASS] testHighExpos() (gas: 567279549)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.63s (1.60s CPU time)
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

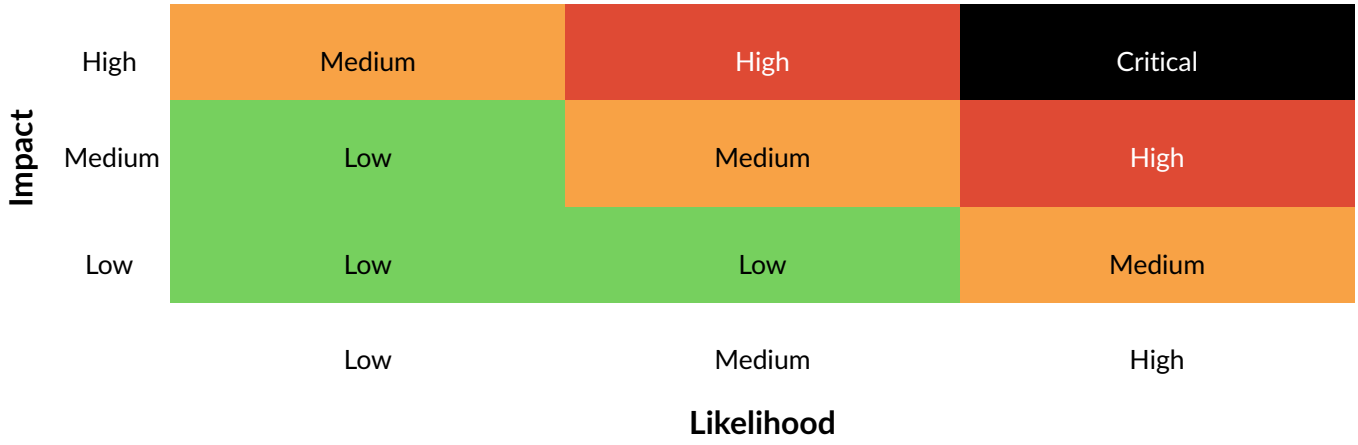

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.