Blackthorn

Security Review For **Ethereum Foundation**

Collaborative Audit Prepared For:	**Ethereum Foundation**
Lead Security Expert(s):	**0xadrii**
	0x52
	hack3r-0m
	merkleplant
Date Audited:	**October 28th - November 2nd**
Final Commit:	**1d679164e03d73dc7f9a5331b67fd51e7032b104**

Introduction

This repository stores geas implementations of Ethereum's system contracts, such as the ones associated with EIP-7002 and EIP-7251.

Scope

Repository: https://github.com/lightclient/sys-asm

Commit: b5b9f33f6b6e7d80f040226e082f74045ddf2c38

Contracts:

- src/consolidations
- src/withdrawals
- src/execution_hash

Final Commit Hash

https://github.com/lightclient/sys-asm/commit/1d679164e03d73dc7f9a5331b67fd51e7032b104

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
0	1	16

Issues Not Fixed or Acknowledged

High	Medium	Low/Info
0	0	0

Security Experts Dedicated to This Review

@0xadrii

@0x52

@hack3r-0m

@merkleplant

Issue M-1: Incorrect check allows returning the same hash for two different blocks in the ring buffer

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/14

Summary

An incorrect check on the lower limit of the ring buffer allows users to retrieve the hash of a block that should no longer be accessible. This results in two different blocks returning the same block hash.

Root Cause

In main.eas#L70, a check is performed to validate whether the input requests a block hash prior to the earliest available. The check intends to ensure `number - input < BUFLEN`, but incorrectly includes an additional +1 to `BUFLEN`:

This mistake permits querying `BUFLEN+1` slots, even though only `BUFLEN` hashes are stored. Due to how data is stored in the ring buffer, requesting the hash for `block.number - 1` and for `block.number - 1 - BUFLEN` will access the same slot without reverting, causing two different blocks to return the same block hash.

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

Medium severity. The contract permits querying a block outside the expected range, returning an incorrect block hash instead of reverting as expected.

PoC

The following Proof of Concept illustrates how querying two different blocks returns the same block hash:

```
function test_FetchLimit() public {
        uint256 start = 8185;
        vm.roll(start);

        // Store hashes for blocks.
        for(uint256 i; i < buflen + 200; i++) {
            vm.prank(sysaddr);
            (bool ret, bytes memory data) =
             ↪  unit.call(abi.encode(keccak256(abi.encode(i))));

            require(ret, "failed");
            assertEq(data, hex"");

            vm.roll(block.number+1);
        }

        // At block 16382, only blocks from [8191, 16381] should be able to be
         ↪  queried.
        vm.roll(16382);

        // However, it is possible to query block 8190, which accesses slot 8190
         ↪  (8190%BUFLEN).
        (bool ret, bytes memory hash) = unit.call(abi.encode(8190));
        require(ret, "Failed fetching");
        console.logBytes(hash);

        // Which shows the same result as querying 16381, which also accesses slot
         ↪  8190 (16381%BUFLEN).
```

```
        (ret, hash) = unit.call(abi.encode(16381));
        require(ret, "Failed fetching");
        console.logBytes(hash);

    }
```

Mitigation

```
  ;; Check if the input is requesting a block hash before the earliest available
  ;; hash currently. Since we've verfied that input <= number - 1, it's safe to
  ;; check the following:
  ;; number - 1 - input <= BUFLEN, which also equals: number - input < BUFLEN
  dup1                ;; [input, input]
-  number             ;; [number, input, input]
-  sub                ;; [number - input, input]
-  push BUFLEN+1      ;; [buflen, number - input, input]
-  lt                 ;; [buflen < number - input, input]
+ push BUFLEN         ;; [buflen, input, input]
+ number              ;; [number, buflen, input, input]
+ sub                 ;; [number-buflen, input, input]
+ swap1               ;; [input, number-buflen, input]
+ lt                  ;; [input< number - buflen, input]
 jumpi @throw         ;; [input]
```

Discussion

lightclient

fixed in https://github.com/lightclient/sys-asm/pull/35/commits/6cae17ba475d23a4cbafb99d379e76793954902b

0xadrii

Fix confirmed

Issue L-1: Use MSIZE instead of PUSH32

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/17

The protocol team has acknowledged this issue.

Summary

The codebase contains multiple instances where push 32 is used to specify memory size for return operations of 32-byte values.

When returning a single 32-byte value that was just stored to memory, using msize instead would be more gas efficient as it avoids pushing an additional value to the stack. This optimization would save 1 gas per occurrence.

- https://github.com/sherlock-audit/2024-10-ethereum-foundation/blob/main/sys-asm/src/execution_hash/main.eas#L83
- https://github.com/sherlock-audit/2024-10-ethereum-foundation/blob/main/sys-asm/src/withdrawals/main.eas#L178
- https://github.com/sherlock-audit/2024-10-ethereum-foundation/blob/main/sys-asm/src/consolidations/main.eas#L183

Root Cause

No response

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

No response

PoC

No response

Mitigation

```
- push 32                ;; [32]
+ msize                  ;; [32]
```

Discussion

lightclient

Given the complexity of these contracts already, I am hesitant to rely on the size of the memory for such small savings.

Issue L-2: Incorrect stack comment in EIP-7251 addresses last 32 bytes of `target` as `target[16:32]`, instead of `target[16:48]`

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/16

Summary

In these stack comments from the consolidation contract, the last 32 bytes of `target` are labeled as `target[16:32]`. However, target is a 48-byte value, with the first 16 bytes (`target[0:16]`) stored in slot 2, and the last 32 bytes (`target[16:48]`) stored in slot 3. This makes the `target` references in the stack comments inaccurate.

Root Cause

No response

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

No response

PoC

No response

Mitigation

Update the comments addressing the last 32 bytes in `target` from `target[16:32]` to `target[16:48]`.

Discussion

lightclient

already fixed

0xadrii

Fix confirmed

Issue L-3: Incorrect comment in EIP-2935 mentions roots instead of block hashes

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/15

Summary

The withdrawals contract contains the following comment:

This comment is incorrect, as in the context of withdrawals and the ring buffer, ”roots” are not read; instead, block hashes are pulled.

Root Cause

No response

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

No response

PoC

No response

Mitigation

Update the comment to the following:

```
...
.start:
  ;; Protect the submit routine by verifying the caller is equal to
  ;; sysaddr().
  caller            ;; [caller]
  push SYSADDR      ;; [sysaddr, caller]
  eq                ;; [sysaddr == caller]
  jumpi @submit     ;; []

  ;; Fallthrough if addresses don't match -- this means the caller intends
-  ;; to read a root.
+  ;; to read a block hash.
...
```

Discussion

lightclient

fixed here https://github.com/lightclient/sys-asm/pull/35/commits/24bb38cdeb7e96e77d912f241234aa986e4fed32

0xadrii

Fix confirmed

Issue L-4: Update `counter` semantics for optimized gas usage in consolidations and withdrawals

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/13

Summary

The `counter` variable at slot 1 in the withdrawals and consolidations contract is used to track the number of new requests in the current block. Users adding a new request increase the counter by one, and the `sysaddr` uses the counter to compute the new `excess` value and afterwards resets the slot to zero.

In order to optimize the gas usage for the first user of a block adding a new request the counter semantics could be updated to be ”off by one” to prevent the user paying the high gas cost of writing to a zero slot. This would change the gas cost for the `counter` update from 22,100 to only 5,000 (assuming slot is cold).

Root Cause

No response

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

No response

PoC

No response

Mitigation

The updated semantics can be easily contained in two additional macros used during the `sysaddr`'s execution path. A user's execution path does not need to be updated as the `counter` is only incremented.

In the `skip_reset` code block the following lines can be updated to use a `read_counter` macro:

```
skip_reset:
  push SLOT_COUNT       // [count_slot, excess, count]
  sload                 // [count, excess, count]
```

Could be updated to:

```
skip_reset:
  %read_counter()       // [count, ...]

// ...

#define %read_counter() {
    push 1              // [1]
    push SLOT_COUNT     // [count_slot, 1]
    sload               // [count, 1]
    sub                 // [count - 1]
}
```

Afterwards, the zeroing of the `counter` slot needs to be updated from:

```
// Reset withdrawal request count.
push 0                // [0, count]
```

```
push SLOT_COUNT       // [count_slot, 0, count]
sstore                // [count]
```

to:

```
%reset_counter()

// ...

#define %reset_counter() {
    push 1            // [1]
    push SLOT_COUNT   // [count_slot, 1]
    sstore            // []
}
```

Note that the change of semantics is solely encapsulated in the macro at which it can be documented. No additional documentation is needed when the macros are used.

Discussion

lightclient

We will consider this, however it is not likely the current gas semantics will always hold true. So we may add complexity to the contract which will not serve the expected purpose in the future.

Issue L-5: Incorrect stack comments in withdrawals and consolidations `read_requests` code block

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/12

Summary

Both system contracts contain the following code block in their `read_requests` path, see consolidations/main.eas:211-216 and withdrawals/main.eas:206-211:

```
read_requests:
  // Determine the size of the queue by calculating tail - head.
  push QUEUE_TAIL       // [tail_idx_slot, head_idx, head_idx]
  sload                 // [tail_idx]
  push QUEUE_HEAD       // [head_idx_slot, tail_idx]
  sload                 // [head_idx, tail_idx]
```

In both cases the `push QUEUE_TAIL` stack comment contains two unaccounted values, `head_idx` and `head_idx`.

Root Cause

No response

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

No response

PoC

No response

Mitigation

Remove the last two elements of the `push QUEUE_TAIL` stack comment.

Discussion

lightclient

fixed in https://github.com/lightclient/sys-asm/pull/35/commits/5b917941657b1fe52ae02e850d12544cfeaae51f

pmerkleplant

Fix confirmed

Issue L-6: Incorrect stack comment in consolidation's `accum_loop` code block

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/11

Summary

The following code block in consolidations/main.eas:290-297 contains faulty stack comments (note that `;;` is updated to `//` to enable nicer syntax support):

```
// @audit Line with last correct stack comments.
mul                  // [offset, target[16:32], src[32:48] ++ tgt[0:16],
↪  source[0:32], addr, i, ..]

// Shift addr bytes.
swap4                // [addr, src[32:48] ++ tgt[0:16], source[0:32],
↪  target[16:32], offset, i, ..]
push 12*8            // [96, addr, src[32:48] ++ tgt[0:16], source[0:32],
↪  target[16:32], offset, i, ..]
shl                  // [addr<<96, src[32:48] ++ tgt[0:16], source[0:32],
↪  target[16:32], offset, i, ..]

// Store addr at offset = i*RECORD_SIZE.
dup5                 // [offset, addr<<96, offset, src[32:48] ++ tgt[0:16],
↪  source[0:32], target[16:32], i, ..]
mstore               // [offset, src[32:48] ++ tgt[0:16], source[0:32],
↪  target[16:32], i, ..]

// @audit Line with first correct stack comments.
// Store source[0:32] at offset = i*RECORD_SIZE + 20.
swap2                // [source[0:32], src[32:48] ++ tgt[0:16], target[16:32],
↪  offset, i, ..]
```

The stack comment of the `swap4` operation should be `[addr, target[16:32], src[32:48] ++ tgt[0:16], source[0:32], offset, i, ..]`. This error continues until the `swap2` operation at which the stack comments are correct again.

Root Cause

No response

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

No response

PoC

No response

Mitigation

Update the stack comments accordingly.

Discussion

lightclient

already fixed

Issue L-7: The `fake_expo` functionality is unhygienic as it leaves intermediate values on the stack

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/10

Summary

The `fake_expo` functionality used in the consolidations and withdrawals function is used to compute the request fee given current `excess` value. However, the ”function” leaves, additionally to the `fee` value, the intermediate values `i`, `numer`, and `accum` on the stack. These values are not documented in the respective system contracts' stack comments.

Root Cause

No response

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

No response

PoC

No response

Mitigation

In order to document the correct state of the stack in the system contracts the additional values may either be documented in the stack comments or the `fake_expo` "function" may `pop` those values of the stack.

However, no issues were found due to the undocumented trailing stack elements.

Discussion

lightclient

fixed in https://github.com/lightclient/sys-asm/pull/35/commits/d4fed45f2989ac80dd85e6d016e30842688e1868

pmerkleplant

Fix confirmed

Issue L-8: Overflow in `fake_expo` leads to spec mismatch

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/9

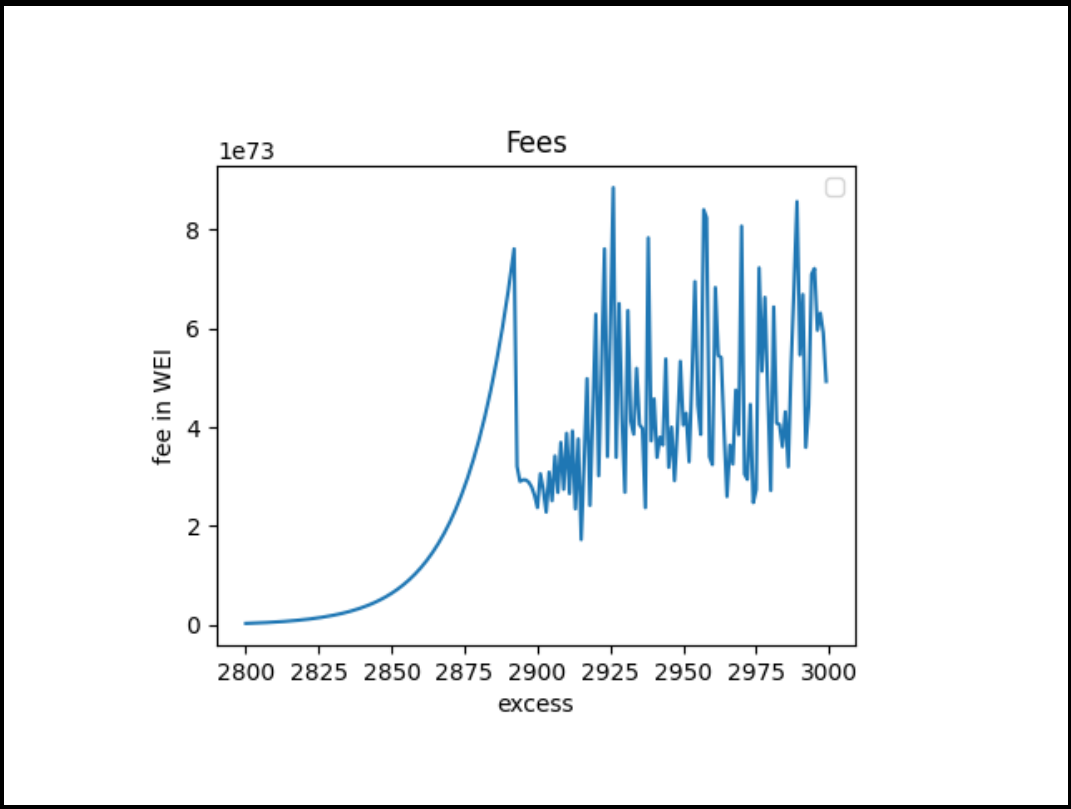
Summary

The `fake_exponential` function used in the consolidations and withdrawals contracts does not conform to the EIP's python specification for values of `excess > 2892`.

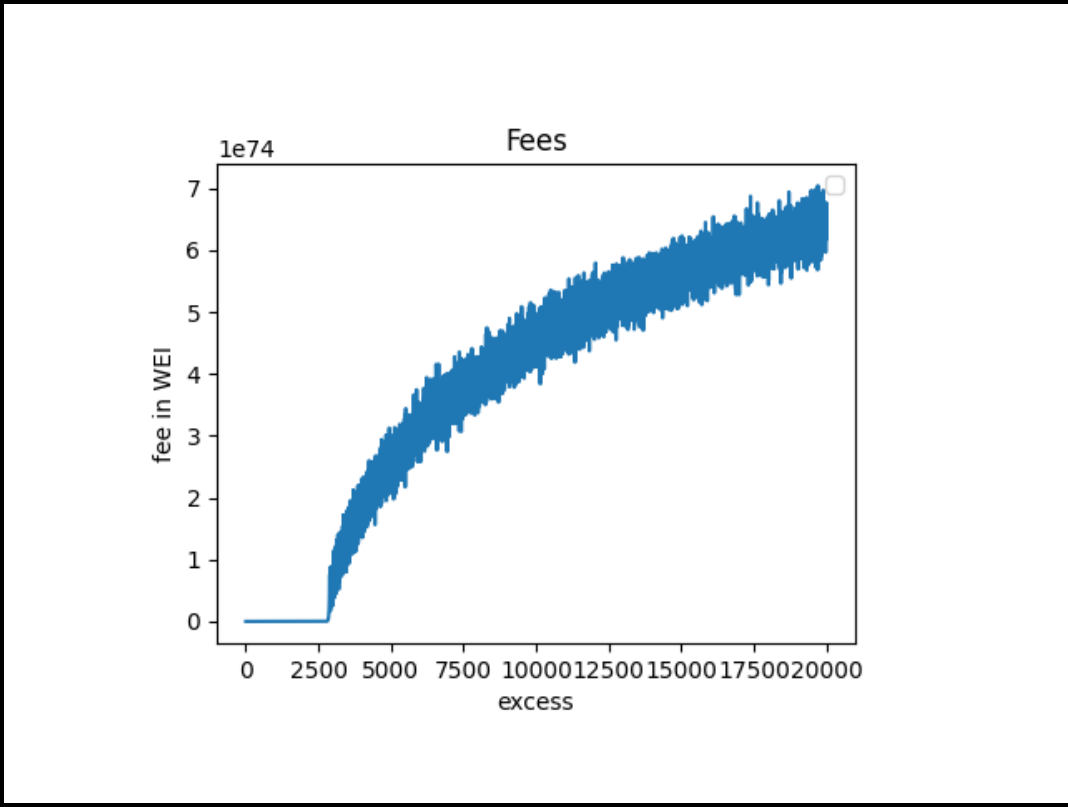
While it is important to note that under current settings it is impossible to reach such a high `excess` value, it may be possible in the future assuming heavy scalability increases. For more info see EIP-7251, 7002 DoS Analysis.

The issue lies in the fact that the geas implementation uses native `uint256` values which overflow for `excess > 2892`. This breaks an important implicit invariant of the fee mechanism, namely an increasing `excess` value leads to an increasing fee.

Plotting the resulting fees for `excess` in the range of `[2800, 3000]` shows that this invariant is broken:



However, note that the functions behaviour stays reasonable though, ie the fee stays incredibly high and tends to increase long term:



Root Cause

No response

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

No response

PoC

No response

Mitigation

No mitigation necessary.

Issue L-9: EIP-{7251, 7002} DoS Analysis

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/8

Summary

The consolidations and withdrawals system contracts both incorporate a fee mechanism for requests to prevent spamming. Without the fee mechanism it would be possible to DoS the contracts with requests to prevent honest users' requests from being processed in an adequate amount of time.

Note that while DoS-ing the consolidations contract can only be attributed to griefing, delaying processing of honest withdrawal requests may lead to financial damage for users, and thereby potential financial gains for staking competitors, as validator payouts can be delayed. However, note that ”sweeping” validator withdrawals by the consensus layer takes multiple days already anyway.

Both fee mechanisms are based on the `fake_exponential` function to increase a requests fee exponentially whenever the amount of requests in a block is higher than the contract's `TARGET`. Its important to note that fees are only updated *after* a block, ie fees are constant for the duration of a block.

The consolidations and withdrawals fee mechanisms' differ in the following ways:

- For consolidations, the `TARGET` requests per block is 1, for withdrawals its 2
- For consolidations, the number of requests processed per block is 1, for withdrawals its 16

In this analysis the following values are used:

- The `GAS_COST` to add a request is underestimated from ~115,000 gas to 100,000 gas
- A block's `GAS_LIMIT`, currently sitting at 30MM, is heavily overestimated by 10x to 300MM

In order to most effectively delay honest requests an attacker would want to add as many requests as possible per block to prevent fee updates. The amount of requests possible in a single block is therefore below `MAX_REQUESTS = GAS_LIMIT / GAS_COST = 3,000`. Assuming an `excess` value of zero this leads to a total fee of only 3,000 wei,

meaning the cost of the attack is solely the opportunity cost of the waived execution fees and mev rewards for the proposed block.

To process 3,000 requests takes (assuming a block time of 12s):

- For consolidations: `MAX_REQUESTS / 1 = 3,000` blocks, ie 10 hours
- For withdrawals: `MAX_REQUESTS / 16 = 188` blocks, ie ~0.6 hours

Continuing the attack in the next block is impossible due to the updated fee reaching a value > `2^250`. However, this means the attacked contract is still DoSed as honest users cannot add new requests.

For the fee to reach < `1 ETH` again it takes:

- For consolidations ~7.6 hours
- For withdrawals ~3.8 hours

To summaries, even with highly beneficial estimations for the attacker, honest user requests cannot reasonably be delayed for longer than 10 hours for consolidations, and ~7.6 hours for withdrawals.

Root Cause

No response

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

No response

PoC

No response

Mitigation

No response

Discussion

lightclient

this is useful, thank you. not sure if there is anything further to update here.

Issue L-10: Comment confuses stack and storage in EIP-{7251, 7002} system contracts

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/7

Summary

Both contracts zero the `excess` value if `excess == INHIBITOR`, see consolidations/main.eas:369-371 and withdrawals/main.eas:378-379.

In order to do so the `excess` value currently on the stack is dropped and substituted by zero, while the comment states: `Drop the excess from storage and use 0.`.

Root Cause

No response

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

No response

PoC

No response

Mitigation

Update the comments to `Drop the excess from stack and use 0.`.

Discussion

lightclient

fixed in https://github.com/lightclient/sys-asm/pull/35/commits/c2c223b8024808f553089a9c53ad1f1217cd3d64

Issue L-11: Unnecessary `iszero(iszero())` in EIP-{7251, 7002} system contracts

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/6

Summary

In both system contracts the following code block is executed to return the current `excess` value:

```
read_excess:
  ;; This is the read path, where we return the current excess.
  ;; Reject any callvalue here to prevent lost funds.
  callvalue             ;; [value]
  iszero                ;; [value == 0]
  iszero                ;; [value != 0]
  jumpi @revert
```

The two `iszero` opcodes are unnecessary as the jump will only be executed if `callvalue != 0` anyway.

Root Cause

No response

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

No response

PoC

No response

Mitigation

No response

Discussion

lightclient

already fixed

Issue L-12: Incorrect comment in EIP-{7251, 7002} system contracts

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/5

Summary

In consolidations/main.eas:55-56 the comment states: `This is the default code path. It will attempt to record a user's request so long as they pay the required fee.`. The same comment is in withdrawals/main.eas:60-61.

However, in both contracts there are still two code paths open, one for adding a user request and a second for reading the `excess` value.

Root Cause

No response

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

No response

PoC

No response

Mitigation

Move the comment below the next code block which branches the execution to reading the `excess` value if no calldata is provided.

Discussion

lightclient

already fixed

Issue L-13: Use `foundry`'s ffi capabilities to compile geas code

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/4

Summary

Foundry offers the `vm.ffi()` function to call external programs. This functionality can be used to compile the system contracts directly in the tests' `setUp()` function, enabling the removal of the `build-wrapper` script.

Root Cause

No response

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

No response

PoC

No response

Mitigation

Implement a `Geas.sol` library similar to:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Vm} from "forge-std/Vm.sol";

library Geas {
    Vm private constant vm = Vm(address(uint160(uint(keccak256("hevm cheat
    ↪  code")))));

    function compile(string memory path) internal returns (bytes memory) {
        string[] memory args = new string[](3);
        args[0] = "geas";
        args[1] = "-no-nl";
        args[2] = path;

        return vm.ffi(args);
    }
}
```

The library can then be used in the `setUp()` functions:

```
function setUp() public {
    vm.etch(addr, Geas.compile("src/execution_hash/main.eas"));
}
```

Afterwards either add `ffi = true` to the `foundry.toml` file or run tests with the `--ffi` flag, eg `forge t --ffi`.

Discussion

lightclient

done with

https://github.com/lightclient/geas-ffi https://github.com/lightclient/sys-asm/pull/35

/commits/76a281763aca2ad495ac9669128fe8f5c31817c2

pmerkleplant

Fix confirmed

Issue L-14: Comment is off by one in EIP-2935's ring buffer implementation

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/3

Summary

In execution_hash/main.eas:52-53 the comment states: `Check if input is requesting a block hash greater than current block number.`.

This comment is off by one as the actual check is whether the block hash is greater than current block number - 1.

Root Cause

No response

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

No response

PoC

No response

Mitigation

Update the comment to: `Check if input is requesting a block hash greater than current block number - 1.`

Discussion

lightclient

already fixed

Issue L-15: Negate user's `calldatasize` revert condition in EIP-2935's ring buffer

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/2

Summary

The execution hashes ring buffer implementation has three revert conditions for non-`sysaddr` callers:

- `calldatasize != 32`
- `calldatasize(0)` outside of upper buffer bound
- `calldatasize(0)` outside of lower buffer bound

While a failure for the last two conditions leads to a `jumpi` to a reverting code block, the first condition is implemented such that the happy path does the `jumpi`. Note that this leads to two distinct reverting code blocks in the contract.

In order to optimize the happy path, and reduce bytecode size, the first condition may be negated via `sub(calldatasize(), 32)` leading to the reverting path doing the `jumpi`. Note that this also enables removing the second reverting code block leading to a bytecode decrease of 5 bytes.

Root Cause

No response

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

No response

PoC

No response

Mitigation

1. Update the `calldatasize` check from `calldatasize() == 32` to `calldatasize() - 32`
2. Update `jumpi @load` to `jumpi @throw`
3. Remove the `load:` label for the happy path
4. Remove the `pop` opcode from the `throw:` label code block. Note that `pop`-ing a stack value right before reverting is not necessary

Discussion

lightclient

this is a nice one

done here https://github.com/lightclient/sys-asm/pull/35/commits/d902117908f07dd4f84fc0f371b9be94f8f5ba46

pmerkleplant

Fix confirmed

Issue L-16: EIP-7002's specification is missing `excess` inhibitor reset

Source: https://github.com/sherlock-audit/2024-10-ethereum-foundation/issues/1

Summary

At construction the `excess` value is set to `type(uint).max`, called *inhibitor*, to indicate the contract is not yet usable. During the first call of the `sysaddr`, ie at the start of the fork activation block, `excess` is updated to zero to activate the contract.

The zeroing of the `excess` value, if and only if `excess == type(uint).max`, is missing in the EIP's specification.

Root Cause

The EIP's `update_excess_withdrawal_requests()` function is missing the following lines of code directly after reading the `previous_excess` value:

```
# Check if excess needs to be reset to 0 for first iteration after activation
if previous_excess == EXCESS_INHIBITOR:
    previous_excess = 0
```

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

No response

Impact

No response

PoC

No response

Mitigation

Inside EIP-7002, update the `update_excess_withdrawal_requests()` to:

```
def update_excess_withdrawal_requests():
    previous_excess = sload(WITHDRAWAL_REQUEST_PREDEPLOY_ADDRESS,
    ↪  EXCESS_WITHDRAWAL_REQUESTS_STORAGE_SLOT)
    # Check if excess needs to be reset to 0 for first iteration after activation
    if previous_excess == EXCESS_INHIBITOR:
        previous_excess = 0

    count = sload(WITHDRAWAL_REQUEST_PREDEPLOY_ADDRESS,
    ↪  WITHDRAWAL_REQUEST_COUNT_STORAGE_SLOT)

    new_excess = 0
    if previous_excess + count > TARGET_WITHDRAWAL_REQUESTS_PER_BLOCK:
        new_excess = previous_excess + count - TARGET_WITHDRAWAL_REQUESTS_PER_BLOCK

    sstore(WITHDRAWAL_REQUEST_PREDEPLOY_ADDRESS,
    ↪  EXCESS_WITHDRAWAL_REQUESTS_STORAGE_SLOT, new_excess)
```

Discussion

lightclient

fixed in https://github.com/ethereum/EIPs/pull/9040/commits/c08f7cbf1a439ee5e94d2294c21e3c13ac2f514c

pmerkleplant

Fix confirmed

Disclaimers

Blackthorn does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.