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	0	2

Issues Not Fixed or Acknowledged

High	Medium	Low/Info
0	0	0

Security Experts Dedicated to This Review

@0xadrii

[illegible]

@0x52

[illegible]

@hack3r-0m

[illegible]

@merkleplant

[illegible]

Issue L-1: 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-2: Overflow in `fake_expo` leads to spec mismatch

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

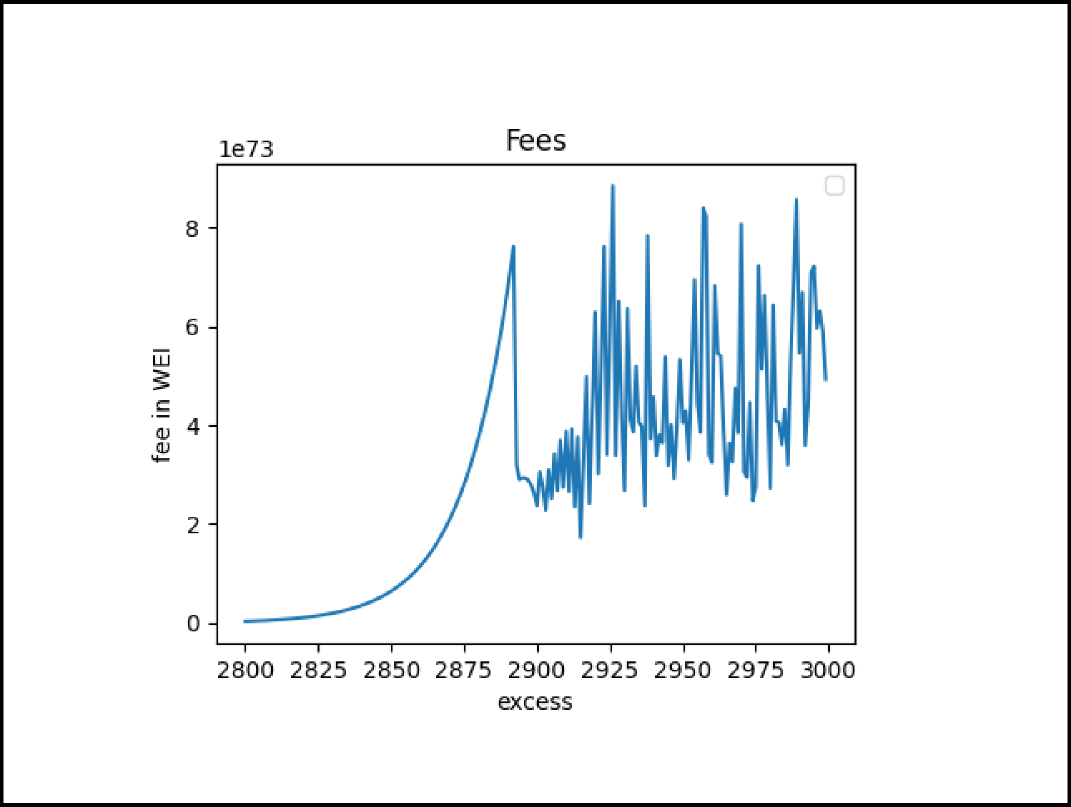
Summary

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`.

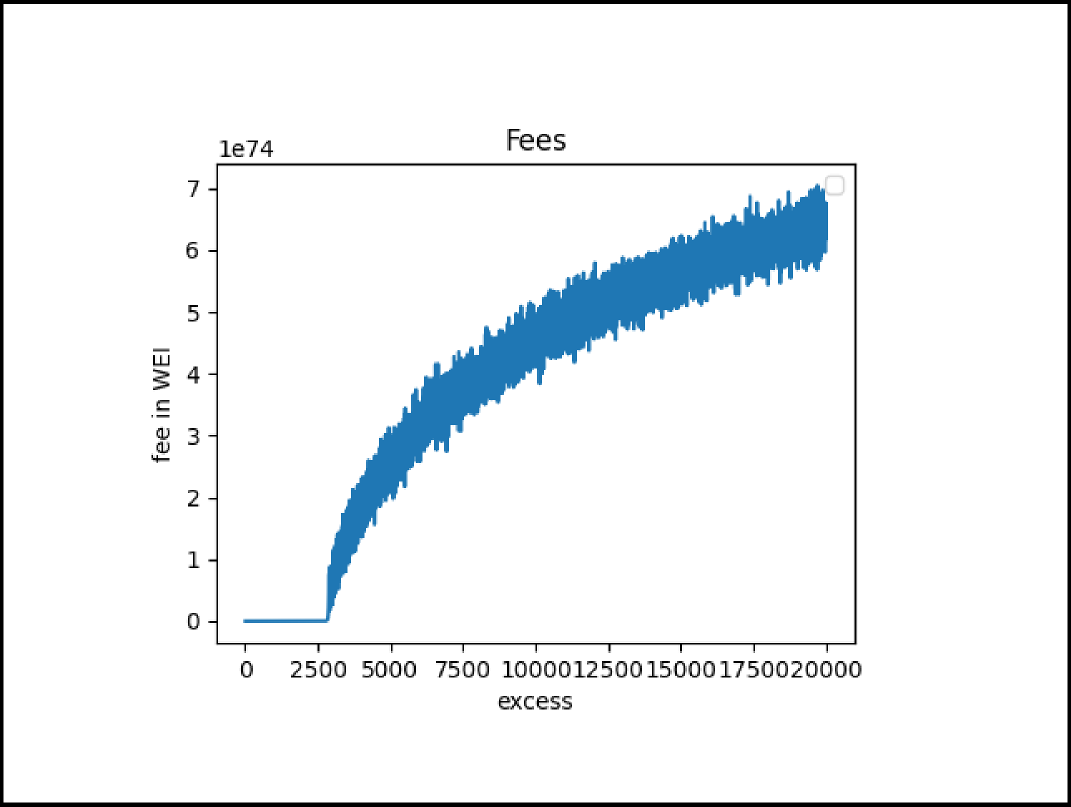
While it is important to note that under current settings it is impossible to reach such a high `excess` value, it may be possible in the future assuming heavy scalability increases. For more info see EIP-7251, 7002 DoS Analysis.

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.

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.