CANTINA

OpenVM v1.5.0

Security Review

Cantina Managed review by:

Zigtur, Lead Security Researcher

Om Parikh, Security Researcher

January 23, 2026

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

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it 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 that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

OpenVM is a performant and modular zkVM framework built for customization and extensibility.

From Jan 5th to Jan 9th the Cantina team conducted a review of stark-backend, Plonky3 and openvm on commit hashes 222e5df3, 1eed8e6d and b4d5716b. The team identified a total of **6** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	1	1	0
Gas Optimizations	0	0	0
Informational	5	5	0
Total	**6**	**6**	**0**

2.1 Scope

The security review had the following components in scope:

```
├── openvm
│   ├── benchmarks
│   │   ├── guest
│   │   │   └── kitchen-sink
│   │   │       ├── openvm_init.rs
│   │   │       └── src
│   │   │           └── main.rs
│   │   └── prove
│   │       └── src
│   │           ├── bin
│   │           │   └── verify_fibair.rs
│   │           └── util.rs
│   ├── crates
│   │   ├── cli
│   │   │   └── src
│   │   │       └── default.rs
│   │   └── sdk
│   │       └── src
│   │           ├── config
│   │           │   └── mod.rs
│   │           ├── keygen
│   │           │   ├── dummy.rs
│   │           │   └── mod.rs
│   │           └── prover
│   │               └── agg.rs
│   └── extensions
│       └── native
│           ├── circuit
│           │   └── src
│           │       └── extension
│           │           └── mod.rs
│           └── recursion
│               └── src
│                   ├── challenger
│                   │   └── duplex.rs
│                   └── fri
│                       ├── hints.rs
│
```

```
│                              ├── two_adic_pcs.rs
│                              ├── types.rs
│                              └── witness.rs
├── Plonky3
│   ├── field
│   │   └── src
│   │       └── field.rs
│   ├── fri
│   │   └── src
│   │       └── fri.rs
│   └── keccak-air
│       └── src
│           ├── air.rs
│           └── round_flags.rs
└── stark-backend
    └── crates
        └── stark-sdk
            └── src
                └── config
                    └── fri_params.rs
```

3 Findings

3.1 Low Risk

3.1.1 FRI parameters do not strictly provide 100 bits of provable security

Severity: Low Risk

Context: fri_params.rs#L75-L87

Description: The `standard_fri_params_with_100_bits_security` defines FRI parameters to reach 100 bits of provable security. These parameters relies on several assumptions:

- The challenge field has size at least 2^{123}.
- For `log_blowup = 1`, multi-FRI will be run with at most width 30000 at any level.
- For `log_blowup > 1`, multi-FRI will be run with at most width 2000 at any level.

Considering the equation `extension_field_bits - log2(2^max_height * (num_columns - 1))` and the worst-case scenario, we end up with the following calculations.

The OpenVM configuration specifies an extension degree of 4. Since the BabyBear prime is slightly smaller than 2^{31}, the resulting extension field size is strictly less than 124 bits. This satisfies the documented assumption that *"the challenge field has size at least 2^{123}"* (treating 123 bits as the conservative lower bound).

Executing the full calculations results in the query security reaching 100 bits. However, the commit phase security is a bit lower than 100 bits (99.X bits).

Proof of Concept:

```
Using extension field bits: 123.48
======================================================================

--- Analyzing Configuration for log_blowup = 1 ---
[Commit Phase]
  Max Columns:        30,000
  Batching Loss:      39.87 bits (log2(Rows * Cols))
  Base Security:      83.61 bits
  + PoW Grinding:     16 bits
  >> Total Commit Security: 99.61 bits
[Query Phase]
  Code Rate:          0.5
  Unique Decoding:    0.25
  Bits Per Query:     0.4150
  Query Bits:         80.10 bits (193 queries)
  + PoW Grinding:     20 bits
  >> Total Query Security:  100.10 bits

>> FINAL PROVABLE SECURITY: 99.61 bits
--------------------------------------------------

--- Analyzing Configuration for log_blowup = 2 ---
[Commit Phase]
  Max Columns:        2,000
  Batching Loss:      35.97 bits (log2(Rows * Cols))
  Base Security:      87.51 bits
  + PoW Grinding:     12 bits
  >> Total Commit Security: 99.51 bits
[Query Phase]
  Code Rate:          0.25
  Unique Decoding:    0.375
  Bits Per Query:     0.6781
  Query Bits:         80.01 bits (118 queries)
  + PoW Grinding:     20 bits
  >> Total Query Security:  100.01 bits

>> FINAL PROVABLE SECURITY: 99.51 bits
```

```
--------------------------------------------------

--- Analyzing Configuration for log_blowup = 4 ---
[Commit Phase]
  Max Columns:        2,000
  Batching Loss:      35.97 bits (log2(Rows * Cols))
  Base Security:      87.51 bits
  + PoW Grinding:     12 bits
  >> Total Commit Security: 99.51 bits
[Query Phase]
  Code Rate:          0.0625
  Unique Decoding:    0.46875
  Bits Per Query:     0.9125
  Query Bits:         80.30 bits (88 queries)
  + PoW Grinding:     20 bits
  >> Total Query Security:  100.30 bits

>> FINAL PROVABLE SECURITY: 99.51 bits
--------------------------------------------------
```

The following Python script was used for the full calculation.

```python
import math
import argparse

# OpenVM Trace Height Parameters (from codebase analysis):
#
# 1. DEFAULT_MAX_TRACE_HEIGHT_BITS = 22
#    -> Default segment size limit: 2^22 = 4,194,304 rows
#    Found in: openvm/crates/vm/src/arch/execution_mode/metered/segment_ctx.rs
#
# 2. NATIVE_MAX_TRACE_HEIGHTS (component-specific limits):
#    -> Maximum: 2^25 = 33,554,432 rows for FriReducedOpeningAir
#    Found in: openvm/extensions/native/circuit/src/extension/mod.rs
#
# 3. Notable component limits:
#    - Program: 2^22 (4,194,304)
#    - FriReducedOpeningAir: 2^25 (33,554,432) <- LARGEST
#    - FieldArithmeticAir: 2^24 (16,777,216)
#    - AccessAdapter: 2^23 (8,388,608)
#
# For security analysis, we use the maximum (2^25) for worst-case calculations.

def calculate_fri_security(log_blowup, num_queries, commit_pow, query_pow,
↪   extension_field_bits):
    print(f"--- Analyzing Configuration for log_blowup = {log_blowup} ---")

    # ==========================================
    # 1. CONSTANTS (Implicit Assumptions)
    # ==========================================
    # Field Size: BabyBear (approx 31 bits) extended to degree 4 (~124 bits)
    # The comment says "at least 2^123", we use 124 for the exact calculation.
    EXTENSION_FIELD_BITS = extension_field_bits

    # Max Trace Height (Rows): From OpenVM codebase
    # - Default: 2^25 (DEFAULT_MAX_TRACE_HEIGHT_BITS = 25)
    # - Maximum component: 2^25 (FriReducedOpeningAir in NATIVE_MAX_TRACE_HEIGHTS)
    # Using the maximum for worst-case security analysis
    MAX_HEIGHT_LOG2 = 25
    MAX_ROWS = 2**MAX_HEIGHT_LOG2

    # Max Columns (Batching): Varies by blowup factor as per dev reply
    if log_blowup == 1:
        MAX_COLS = 30_000
    else:
        MAX_COLS = 2_000
    # ==========================================
    # 2. COMMITMENT PHASE SECURITY (Batching)
    # ==========================================
    # Formula: Security = FieldBits - log2(Rows * Cols) + PoW
    # Note: Developer used (num_columns - 1), but for approx we use num_columns

    batching_loss_bits = math.log2(MAX_ROWS * (MAX_COLS -1))
    base_commit_security = EXTENSION_FIELD_BITS - batching_loss_bits
    total_commit_security = base_commit_security + commit_pow

    print(f"[Commit Phase]")
    print(f"  Max Columns:       {MAX_COLS:,}")
    print(f"  Batching Loss:     {batching_loss_bits:.2f} bits (log2(Rows * Cols))")
    print(f"  Base Security:     {base_commit_security:.2f} bits")
    print(f"  + PoW Grinding:    {commit_pow} bits")
    print(f"  >> Total Commit Security: {total_commit_security:.2f} bits")

    # ==========================================
    # 3. QUERY PHASE SECURITY (Soundness)
    # ==========================================
    # Formula: Bits = -NumQueries * log2(1 - delta) + PoW
    # delta (Unique Decoding Radius) = (1 - Rate) / 2
    # Rate = 2^(-log_blowup)

    rate = 2**(-log_blowup)
    delta = (1.0 - rate) / 2.0

    # Probability of cheating on one query = 1 - delta
    # If delta is 0.25, prob is 0.75. log2(0.75) is negative.
    bits_per_query = -math.log2(1.0 - delta)

    query_security_from_queries = num_queries * bits_per_query
    total_query_security = query_security_from_queries + query_pow

    print(f"[Query Phase]")
    print(f"  Code Rate:         {rate}")
    print(f"  Unique Decoding:   {delta}")
    print(f"  Bits Per Query:    {bits_per_query:.4f}")
    print(f"  Query Bits:        {query_security_from_queries:.2f} bits ({num_queries}
    ↪  queries)")
    print(f"  + PoW Grinding:    {query_pow} bits")
    print(f"  >> Total Query Security:  {total_query_security:.2f} bits")

    # ==========================================
    # 4. CONCLUSION
    # ==========================================
    min_security = min(total_commit_security, total_query_security)
    print(f"\n>> FINAL PROVABLE SECURITY: {min_security:.2f} bits")
    print("-" * 50 + "\n")

# ==========================================
# Run Checks for OpenVM Parameters
# ==========================================

def main():
    parser = argparse.ArgumentParser(description='Calculate FRI security parameters for
    ↪  OpenVM')
    parser.add_argument('--extension-bits', '-e', type=float, default=30.87 * 4,
                        help='Extension field size in bits (default: 30.87 * 4 = 123.48
                        ↪  for BabyBear^4)')
    args = parser.parse_args()

    extension_bits = args.extension_bits
    print(f"Using extension field bits: {extension_bits:.2f}")
    print("=" * 70 + "\n")
    # Case 1: log_blowup = 1 (Wide mode)
    # Params: queries=193, commit_pow=16, query_pow=20
    calculate_fri_security(log_blowup=1, num_queries=193, commit_pow=16, query_pow=20,
                           extension_field_bits=extension_bits)

    # Case 2: log_blowup = 2 (Standard mode)
    # Params: queries=118, commit_pow=12, query_pow=20
    calculate_fri_security(log_blowup=2, num_queries=118, commit_pow=12, query_pow=20,
                           extension_field_bits=extension_bits)

    # Case 3: log_blowup = 4 (High Blowup mode)
    # Params: queries=88, commit_pow=12, query_pow=20
    calculate_fri_security(log_blowup=4, num_queries=88, commit_pow=12, query_pow=20,
                           extension_field_bits=extension_bits)


if __name__ == "__main__":
    main()
```

Recommendation: Review the FRI parameters in `standard_fri_params_with_100_bits_security`. Consider adding a formal calculations as part of the code with an assertion to ensure 100 bits of security.

OpenVM: Fixed. In PR 231, we provide a security calculator to calculate the Fiat-Shamir soundness of the protocol overall. In doing so, we changed the `max_log_domain_size = BabyBear::TWO_ADICITY = 27` to support all trace height, with some adjustments to the parameters to ensure 100 bits.

However in the course of adding the calculator, we realized that our previous calculations to arrive at 100 bits of security did not account for the error contribution from the DEEP-ALI part of the protocol. See Protocol 3 of *"A summary on the FRI low degree test"*, Ulrich Haböck Due to the nature of the DEEP step, the only way to reach 100 bits of security with the current `BabyBear^4` extension field is to introduce a small additional round of grinding prior to sampling of the `zeta` opening point.

- The prover and verifier changes to add this grinding have been added to stark-backend in stark-backend PR 231.
  - Updated max constraint count to support openvm standard config in stark-backend PR 233.
- The recursion circuit changes correspond to it have been added in openvm PR 2361.

Cantina Managed: Fixed. The calculator ensures that 100 bits of security is reached with the FRI parameters configuration. Additionally, the DEEP-ALI bits of security calculations are executed in this calculator.

These DEEP-ALI security calculations led to discovering that a PoW grinding phase was needed to ensure a security of 100 bits. It has been added in the stark-backend repository and related changes have been made in openvm repository.

The DEEP PoW grinding configuration is included in the verification key (`VK`) and verified by the verifier before the `zeta` challenge.

With the new parameters, the total provable security for the worst-case scenario (trace height 2^{27}) is raised to >100 bits, fully resolving the finding.

3.2 Informational

3.2.1 100 bits of provable security calculations could be documented

Severity: Informational

Context: fri_params.rs#L79-L113

Description: The `standard_fri_params_with_100_bits_security` provides parameters to reach 100 bits of provable security.

> Pre-defined FRI parameters with 100 bits of provable security, meaning we do not rely on any conjectures about Reed-Solomon codes (e.g., about proximity gaps) or the ethSTARK Toy Problem Conjecture.

However, these calculations are not documented for each `FriParameters` structure.

Recommendation: For transparency purposes, consider adding a comment that details the calculation to reach 100 bits of provable security for each `FriParameters` configuration. Another solution is to implement a `get_provable_security_bits()` function that calculates the provable security bits with an assertion in `standard_fri_params_with_100_bits_security` to ensure that it is greater than or equal to 100. This is similar to the `get_conjectured_security_bits` function and its assertion in `standard_fri_params_with_100_bits_conjectured_security`.

OpenVM: Fixed in PR 231

Cantina Managed: Fixed. The `test_params_provable_security` test has been added. It details all security bits calculations and assert that the result is greater than or equal to `100`.

3.2.2 `stark-backend` and `openvm` versions are not correct

Severity: Informational

Context: Cargo.toml#L2, Cargo.toml#L1-L2

Description: The `openvm/Cargo.toml` file shows version `1.4.2` instead of `1.5.0`. Also, the `stark-backend/Cargo.toml` file shows version `1.2.2` instead of `1.3.0`.

Recommendation: Fix the versions for `openvm` and `stark-backend`.

OpenVM: Fixed, stark-backend version set to `v1.3.0-rc.0` in PR 229 and openvm version set to `v1.5.0-rc.0` in PR 231. We will move out of `-rc` prior to the final release.

Cantina Managed: Fixed. `v1.3.0-rc.0` and `v1.5.0-rc.0` are set in the `Cargo.toml` files.

3.2.3 Fibonacci E2E benchmark is not working by default

Severity: Informational

Context: fib_e2e.rs#L14-L16

Description: The Fibonacci E2E benchmark binary panics when it is executed in debug mode. This is because an assertion is executed to ensure that `max_trace_height` is a power of two, however its default value is `1_000_000` which is not a power of two.

Proof of Concept: Run the `fib_e2e` benchmark without any argument.

```
cargo run --package openvm-benchmarks-prove --bin fib_e2e
```

A `debug_assert!` is triggered because the `max_trace_height` is not a power of two.

```
thread 'main' panicked at crates/vm/src/arch/execution_mode/metered/segment_ctx.rs:54:9:
max_trace_height should be a power of two
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

Recommendation: Modify the default `max_segment_length` to be a power of two:

```diff
  diff --git a/benchmarks/prove/src/bin/fib_e2e.rs b/benchmarks/prove/src/bin/fib_e2e.rs
  index aa01c90f4..dfbc1c0ea 100644
-  -- a/benchmarks/prove/src/bin/fib_e2e.rs
+  ++ b/benchmarks/prove/src/bin/fib_e2e.rs
  @@ -12,7 +12,7 @@ async fn main() -> Result<()> {
       let args = BenchmarkCli::parse();

       // Must be larger than RangeTupleCheckerAir.height == 524288
-      let max_segment_length = args.max_segment_length.unwrap_or(1_000_000);
+      let max_segment_length = args.max_segment_length.unwrap_or(1_048_576);
```

```
        let mut config =
            SdkVmConfig::from_toml(include_str!("../../../guest/fibonacci/openvm.toml"))?
            .app_vm_config;
```

OpenVM: Fixed in PR 2364 by using `1 << 20` instead of `1_000_000`.

Cantina Managed: Fixed. The Fibonacci benchmark does not panic anymore.

3.2.4 `todo!` can be changed to more appropriate alternative

Severity: Informational

Context: fri_params.rs#L109

Description: `todo!` is used inside of match statement which indicates not yet complete while the current implementation is complete as-is.

Recommendation: though the runtime behaviour is same, It should be replaced with more appropriate alternative:

- `panic!` → for explicit error.
- `unreachable!` → if inputs are sanitized elsewhere.

OpenVM: Fixed in PR 232.

Cantina Managed: Fix verified.

3.2.5 Kitchen-sink benchmark panics

Severity: Informational

Context: memory_ctx.rs#L217-L225

Description: The kitchen-sink benchmark is meant to be a broad, integration-style workload that uses all zkVM functionalities. With this benchmark, OpenVM's proving/aggregation stack is exercised with all major chips/extensions at once. This benchmark is supposed to be runnable in both `release` and `debug` mode. However, both execution panics in the current state of the codebase.

Proof of Concept: Execute the benchmark in `release` mode.

```
cargo run --package openvm-benchmarks-prove --bin kitchen_sink --release
```

The `release` mode execution ends up panicking with the following logs:

```
openvm build:    Compiling openvm-kitchen-sink-program v0.0.0
    (/.../openvm/benchmarks/guest/kitchen-sink)
[init] complex #0 = Bn254Fp2 (mod_idx = 4)
[init] complex #1 = Bls12_381Fp2 (mod_idx = 6)
openvm build:     Finished `release` profile [optimized] target(s) in 10.17s

thread 'main' panicked at benchmarks/prove/src/bin/kitchen_sink.rs:38:9:
assertion `left == right` failed
  left: 9
 right: 1
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

Execute the benchmark in `debug` mode:

```
cargo run --package openvm-benchmarks-prove --bin kitchen_sink
```

The `debug` mode execution ends up panicking with the following logs:

```
openvm build:    Compiling openvm-kitchen-sink-program v0.0.0
    (/.../openvm/benchmarks/guest/kitchen-sink)
[init] complex #0 = Bn254Fp2 (mod_idx = 4)
[init] complex #1 = Bls12_381Fp2 (mod_idx = 6)
openvm build:     Finished `release` profile [optimized] target(s) in 10.18s
```

```
thread 'main' panicked at
 ↪  /.../openvm/crates/vm/src/arch/execution_mode/metered/memory_ctx.rs:244:9:
align_bits (2) must be <= size_bits (0)
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

Recommendation: Fix the kitchen-sink benchmark.

OpenVM: Fixed. The problems were distinct for debug versus release mode. For release mode: the fix was done in PR 2361. It happened because we extract the max heights in a roundabout way, and we needed to initially turn off segmentation limits (the change was due to an unrelated change in behavior in main). For debug mode, the reason was that some debug info was being lost in between leaf prover runs. That is fixed in PR 2364.

Cantina Managed: Fixed. The kitchen-sink is now runnable in both release and debug mode.