OpenVM Fix Review

Security Review

Solo review by:
Zigtur, Lead Security Researcher

April 23, 2025

Contents

1 Introduction **2**
1.1 About Cantina . . . 2
1.2 Disclaimer . . . 2
1.3 Risk assessment . . . 2
1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
3.1 Fix . . . 4
3.2 Recommendation . . . 4
3.3 Alternative fix . . . 4
3.4 Additional notes . . . 5

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

A security review is 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 the review 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 a 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	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

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 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 Apr 17th to Apr 18th the security researchers conducted a review of openvm's fixes of the security advisory GHSA-jf2r-x3j4-23m7.

The fix implemented for the security advisory GHSA-jf2r-x3j4-23m7 is accurate. It now ensures that the most significant limb is range checked to 6 bits.

3 Findings

```
// pc_limbs[0] is already range checked through rd_data[0]
for (i, limb) in pc_limbs.iter().skip(1).enumerate() {
    if i == pc_limbs.len() - 1 {
```

The previous code was processing `pc_limbs = [0, 1, 2, 3]` in a `for` loop through an iterator. An enumerator was derived from the iterator, but it was skipping the first element before this derivation.

This led to the following processing of `pc_limbs`:

- Round 1: `(i = 0, limb = 1)`.
- Round 2: `(i = 1, limb = 2)`.
- Round 3: `(i = 2, limb = 3)`.

The third round was supposed to trigger the `i == pc_limbs.len() - 1` condition. However, due to the skip being done before enumerating, the condition was never met as `i < pc_limbs.len() - 1` (i.e. `2 < 3`).

This leads to not range checking the last limb to be within 6 bits but range checking to 8 bits.

3.1 Fix

```
// pc_limbs[0] is already range checked through rd_data[0]
for (i, limb) in pc_limbs.iter().enumerate().skip(1) {
    if i == pc_limbs.len() - 1 {
```

The fix derives the enumerator from the iterator before skipping the first element. This leads to the following processing of `pc_limbs`:

- Round 1: `(i = 1, limb = 1)`.
- Round 2: `(i = 2, limb = 2)`.
- Round 3: `(i = 3, limb = 3)`.

Now, the third round will trigger the `i == pc_limbs.len() - 1` condition. This will ensure that the last limb is range checked to be within 6 bits.

Zigtur's comment: The fix is accurate and ensures that the range check will be done on 6 bits for the last limb (index 3).

3.2 Recommendation

Consider adding some comments to explain the exact case:

```
diff --git a/extensions/rv32im/circuit/src/auipc/core.rs b/extensions/rv32im/circuit/src/auipc/core.rs
index b781c63db..c7569b515 100644
--- a/extensions/rv32im/circuit/src/auipc/core.rs
+++ b/extensions/rv32im/circuit/src/auipc/core.rs
@@ -130,7 +130,8 @@ where
         }

         assert_eq!(pc_limbs.len(), RV32_REGISTER_NUM_LIMBS);
-        // pc_limbs[0] is already range checked through rd_data[0]
+        // use enumerate to match pc_limbs[0] => i=0, pc_limbs[1] => i=1, ...
+        // pc_limbs[0] is already range checked through rd_data[0], so we skip it
         for (i, limb) in pc_limbs.iter().enumerate().skip(1) {
             if i == pc_limbs.len() - 1 {
                 // Range check the most significant limb of pc to be in [0,
                 ↪  2^{PC_BITS-(RV32_REGISTER_NUM_LIMBS-1)*RV32_CELL_BITS})
```

3.3 Alternative fix

The issue could have been fixed by checking `pc_limbs.len() - 2`. However, this would be less easily readable:

```
diff --git a/extensions/rv32im/circuit/src/auipc/core.rs b/extensions/rv32im/circuit/src/auipc/core.rs
index b781c63db..66c8826b2 100644
--- a/extensions/rv32im/circuit/src/auipc/core.rs
+++ b/extensions/rv32im/circuit/src/auipc/core.rs
@@ -131,8 +131,8 @@ where

         assert_eq!(pc_limbs.len(), RV32_REGISTER_NUM_LIMBS);
         // pc_limbs[0] is already range checked through rd_data[0]
-        for (i, limb) in pc_limbs.iter().enumerate().skip(1) {
-            if i == pc_limbs.len() - 1 {
+        for (i, limb) in pc_limbs.iter().skip(1).enumerate() {
+            if i == pc_limbs.len() - 2 {
                 // Range check the most significant limb of pc to be in [0,
                 ↪  2^{PC_BITS-(RV32_REGISTER_NUM_LIMBS-1)*RV32_CELL_BITS})
                 need_range_check.push(
                     (*limb).clone()
```

3.4 Additional notes

The finding "PC byte decomposition in AUIPC can overflow" from the Cantina OpenVM competition recommends the following:

> `pc_limbs[1]` should be constrained to be a byte, and `pc_limbs[2]` to 6 bits (similar to the correct byte decomposition in `jalr`).

The fix for this finding was implemented in commit b1536101.