OpenVM v1.2.1 rc.0

Security Review

Cantina Managed review by:

Guido Vranken, Lead Security Researcher

Om Parikh, Security Researcher

Advisor: **Zigtur**, Lead Security Researcher

July 10, 2025

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	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 Jun 3rd to Jun 8th the Cantina team conducted a review of openvm-v1.2.1-rc.0 on commit hash 2c352538. The team identified a total of **5** issues:

Issues Found

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

The Cantina Managed team reviewed OpenVM's `openvm-v1.2.1-rc.0` holistically on commit hash 3c800070 and concluded that all issues were addressed and no new vulnerabilities were identified.

3 Findings

3.1 Medium Risk

3.1.1 `recover_from_prehash` succeeds when it should fail due to missing overflow check

Severity: Medium Risk

Context: ecdsa.rs#L381

Description: The upstream code performs a `checked_add` and returns Error if an overflow occurs:

- recovery.rs#L306-L310

```
Option::<C::Uint>::from(
    C::Uint::decode_field_bytes(&r.to_repr()).checked_add(&C::ORDER),
)
.ok_or_else(Error::new)?
.encode_field_bytes()
```

But openvm does `add_assign` unconditionally.

Proof of Concept:

```
use hex_literal::hex;

use ecdsa::RecoveryId;
use p256::ecdsa::{VerifyingKey, Signature};

/// Signature recovery test vectors
struct RecoveryTestVector {
    pk: [u8; 33],
    msg: [u8; 32],
    sig: [u8; 64],
    recid: u8,
    ok: bool,
}

const RECOVERY_TEST_VECTORS: &[RecoveryTestVector] = &[
    RecoveryTestVector{pk:hex!("020000000000000000000000000000000000000000000000000000000000000000"),msg:hex!(
    ↪  "00000000000000000000FFFFFFFF03030BFFFFFFFFFF030BFFFFFFFFFFFFFF8FC"),sig:hex!("00000000ffffffff00000000
    ↪  00000004319055258e8617b0c46353d039cdaaf0000000000000000000000000000000000000000000000000000000000000
    ↪  01"),recid:2,ok:false},
];


fn main() {
    for vector in RECOVERY_TEST_VECTORS {
        let sig = match Signature::try_from(vector.sig.as_slice()) {
            Ok(_v) => _v,
            Err(_) => {
                assert_eq!(vector.ok, false);
                continue;
            }
        };
        let recid = match RecoveryId::from_byte(vector.recid) {
            Some(_v) => _v,
            None => {
                assert_eq!(vector.ok, false);
                continue;
            }
        };
        let _ = match VerifyingKey::recover_from_prehash(&vector.msg, &sig, recid) {
            Ok(_v) => {
                /* openvm p256 takes this code path */
                _v
            },
            Err(_) => {
                /* Original p256 crate takes this code path */
                assert_eq!(vector.ok, false);
                continue;
            }
        };
    }
}
```

Recommendation: Implement the upstream logic to return error when the addition overflows for full compatibility.

OpenVM: Fixed in PR 1742.

Cantina Managed: Fix verified.

3.1.2 `recover_from_prehash` succeeds when it should fail due to missing public key validation

Severity: Medium Risk

Context: ecdsa.rs#L391

Description: The upstream code returns error if the recovered public key is invalid:

- recovery.rs#L310:

```
let vk = Self::from_affine(pk.into())?;
```

But openvm returns the public key unconditionally.

Proof of Concept:

```
use hex_literal::hex;

use ecdsa::RecoveryId;
use p256::ecdsa::{VerifyingKey, Signature};

/// Signature recovery test vectors
struct RecoveryTestVector {
    pk: [u8; 33],
    msg: [u8; 32],
    sig: [u8; 64],
    recid: u8,
    ok: bool,
}

const RECOVERY_TEST_VECTORS: &[RecoveryTestVector] = &[
    RecoveryTestVector{pk:hex!("020000000000000000000000000000000000000000000000000000000000000000"),msg:hex!(
    ↪  "0000000000000000000000000000000000000000000000000000000CFD5E267CBB5E"),sig:hex!("6b17d1f2e12c4247f8bce6e5
    ↪  63a440f277037d812deb33a0f4a13945d898c2960000000000000000000000000000000000000000000000000cfd5e267cbb
    ↪  5e"),recid:1,ok:false},
];


fn main() {
    for vector in RECOVERY_TEST_VECTORS {
        let sig = match Signature::try_from(vector.sig.as_slice()) {
            Ok(_v) => _v,
            Err(_) => {
                assert_eq!(vector.ok, false);
                continue;
            }
        };
        let recid = match RecoveryId::from_byte(vector.recid) {
            Some(_v) => _v,
            None => {
                assert_eq!(vector.ok, false);
                continue;
            }
        };
        let _ = match VerifyingKey::recover_from_prehash(&vector.msg, &sig, recid) {
            Ok(_v) => {
                /* openvm p256 takes this code path */
                _v
            },
            Err(_) => {
                /* Original p256 crate takes this code path */
                assert_eq!(vector.ok, false);
                continue;
            }
        };
    }
}
```

Recommendation: Return error when the recovered public key is invalid for full compatibility.

OpenVM: Fixed in PR 1743.

Cantina Managed: Fix verified.

3.1.3 `recover_from_prehash` succeeds when it should fail due to missing ECDSA verification

Severity: Medium Risk

Context: ecdsa.rs#L392

Description: The upstream code returns error if the recovered public key and signature combination does not pass ECDSA verification.

- recovery.rs#L312-L313:

```
// Ensure signature verifies with the recovered key
vk.verify_prehash(prehash, signature)?;
```

Proof of Concept:

```
use hex_literal::hex;

use ecdsa::RecoveryId;
use k256::ecdsa::{VerifyingKey, Signature};

/// Signature recovery test vectors
struct RecoveryTestVector {
    pk: [u8; 33],
    msg: [u8; 32],
    sig: [u8; 64],
    recid: u8,
    ok: bool,
}

const RECOVERY_TEST_VECTORS: &[RecoveryTestVector] = &[
    RecoveryTestVector{pk:hex!("020000000000000000000000000000000000000000000000000000000000000000"),msg:hex!(
    ↪  "0000000000000000000000000000000000000000000000000000000000000000"),sig:hex!("00000000000000000000000
    ↪  0000000000000000000000000000000000000001fffffffffbfffffffffffffffeffbaffaeff6f7000000100000000dbd03641
    ↪  40"),recid:1,ok:false},
];


fn main() {
    for vector in RECOVERY_TEST_VECTORS {
        let mut sig = match Signature::try_from(vector.sig.as_slice()) {
            Ok(_v) => _v,
            Err(_) => {
                assert_eq!(vector.ok, false);
                continue;
            }
        };
        let mut recid = vector.recid;

        /* Change false to true to make recover_from_prehash in k256 crate */
        if false {
            if let Some(sig_normalized) = sig.normalize_s() {
                sig = sig_normalized;
                recid ^= 1;
            }
        }

        let recid = match RecoveryId::from_byte(recid) {
            Some(_v) => _v,
            None => {
                assert_eq!(vector.ok, false);
                continue;
            }
        };
        let _ = match VerifyingKey::recover_from_prehash(&vector.msg, &sig, recid) {
            Ok(_v) => {
                println!("recover_from_prehash: ok");
                /* openvm k256 takes this code path */
                _v
```

```
            },
            Err(_) => {
                println!("recover_from_prehash: err");
                /* k256 crate takes this code path */
                assert_eq!(vector.ok, false);
                continue;
            }
        };
    }
}
```

Recommendation: Return error when the recovered public key and signature combination does not pass ECDSA verification for full compatibility.

OpenVM: The discrepancy is actually because `k256` has an extra step in their `verify_prehash` that forces the signature to be normalized. We address this in PR 1744 and give a proof that the rest of the ECDSA signature verification is automatic from a successful public key recovery.

Cantina Managed: Fix verified.

3.1.4 `VerifyingKey::from_sec1_bytes` panics when parsing unreduced coordinates

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: If `VerifyingKey::from_sec1_bytes` is called when a point representation whose coordinate(s) are equal or greater than the curve prime, a panic (assert failure) will occur due to field arithmetic assuming reduced input:

- is_eq.rs#L349.
- weierstrass.rs#L94.

The proof of concept demonstrates this for `k256` but it applies to `p256` as well.

Proof of Concept:

```
use hex_literal::hex;
use k256::ecdsa::VerifyingKey as VkSecp256k1;

fn main() {
    {
        /* Secp256k1 */

        /* Fails because point is invalid and coordinate (Y) exceeds prime
         *
         * This panics in openvm:
         * thread 'guest_tests::test_pubkey' panicked at /home/jhg/openvm-june/v1.2.1-rc.0/openvm/extensio
         ↪  ns/algebra/circuit/src/modular_chip/is_eq.rs:349:13:
         * [255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
         ↪  0, 0, 0, 255, 255, 255, 255] >= [255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 0, 0,
         ↪  0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 255, 255, 255, 255]
         */
        assert!(
            VkSecp256k1::from_sec1_bytes(
                &hex!("04"
                      "0000000000000000000000000000000000000000000000000000000000000000"
                      "fffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f")
                ).is_err()
            );

        /* Succeeds because it is a valid point
         * openvm behavior is equivalent (e.g. no bug)
         */
        assert!(
            VkSecp256k1::from_sec1_bytes(
                &hex!("04"
                      "0000000000000000000000000000000000000000000000000000000000000001"
                      "4218f20ae6c646b363db68605822fb14264ca8d2587fdd6fbc750d587e76a7ee")
                ).is_ok()
            );

        /* Same point as above but with prime added to X.
```

```
         * Fails because from_sec1_bytes rejects points with oversized coordinates,
         * even if the point is valid after reduction.
         *
         * This also panics in openvm in is_eq.rs.
         */
        assert!(
            VkSecp256k1::from_sec1_bytes(
                &hex!("04"
                      "ffffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc30"
                      "4218f20ae6c646b363db68605822fb14264ca8d2587fdd6fbc750d587e76a7ee")
                ).is_err()
            );
    }
}
```

Recommendation: Reject coordinates >= prime in `from_sec1_bytes` by returning error, and do this before invoking any arithmetic that assumes reduced inputs.

OpenVM: Fixed in PR 1746.

Cantina Managed: Fix verified.

3.1.5 `VerifyingKey::from_sec1_bytes` accepts infinity point whereas upstream does not

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: The infinity point is not a valid `VerifyingKey`. Upstream correctly rejects it whereas openvm does not.

Proof of Concept:

```
use hex_literal::hex;
use k256::ecdsa::VerifyingKey;

fn main() {
    assert_eq!(
        VerifyingKey::from_sec1_bytes(
            &hex!("04"
                  "0000000000000000000000000000000000000000000000000000000000000000"
                  "0000000000000000000000000000000000000000000000000000000000000000")
            ).is_ok(), false);
}
```

Recommendation: Reject the infinity point as a `VerifyingKey`.

OpenVM: Fixed in PR 1747.

Cantina Managed: Fix verified.