hexens x Citrea

Security Review Report for Citrea

April 2025

Table of Contents

1. About Hexens

Hexens is a pioneering cybersecurity firm dedicated to establishing robust security standards for Web3 infrastructure, driving secure mass adoption through innovative protection technology and frameworks. As an industry elite experts in blockchain security, we deliver comprehensive audit solutions across specialized domains, including infrastructure security, Zero Knowledge Proof, novel cryptography, DeFi protocols, and NFTs.

Our methodology combines industry-standard security practices combined with unique methodology of two teams per audit, continuously advancing the field of Web3 security. This innovative approach has earned us recognition from industry leaders.

Since our founding in 2021, we have built an exceptional portfolio of enterprise clients, including major blockchain ecosystems and Web3 platforms.

2. Security Review Details

- **Review Led by**

Hayk Andriasyan, Lead Security Researcher

- **Scope**

The analyzed resources are located on:

🔗 https://github.com/chainwayxyz/risc0-to-bitvm2/tree/main/groth16_proof/circuits

📌 **Commit:** `9aab91ce5b7a3248a5dc43bf6b02aca91b1d288f`

The issues described in this report were fixed in the following commit:

🔗 https://github.com/chainwayxyz/risc0-to-bitvm2/pull/22

📌 **Commit:** `96bab06229196af0ea3aaa91ce417e06697b6b25`

- **Changelog**

21 April 2025	Audit start
28 April 2025	Initial report
05 May 2025	Revision received
05 May 2025	Final report

3. Severity Structure

The vulnerability severity is calculated based on two components:

1. Impact of the vulnerability
2. Probability of the vulnerability

Impact	Probability			
	Rare	Unlikely	Likely	Very likely
Low	Low	Low	Medium	Medium
Medium	Low	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

- **Severity Characteristics**

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical

Vulnerabilities that are highly likely to be exploited and can lead to catastrophic outcomes, such as total loss of protocol funds, unauthorized governance control, or permanent disruption of contract functionality.

High

Vulnerabilities that are likely to be exploited and can cause significant financial losses or severe operational disruptions, such as partial fund theft or temporary asset freezing.

Medium

Vulnerabilities that may be exploited under specific conditions and result in moderate harm, such as operational disruptions or limited financial impact without direct profit to the attacker.

Low

Vulnerabilities with low exploitation likelihood or minimal impact, affecting usability or efficiency but posing no significant security risk.

Informational

Issues that do not pose an immediate security risk but are relevant to best practices, code quality, or potential optimizations.

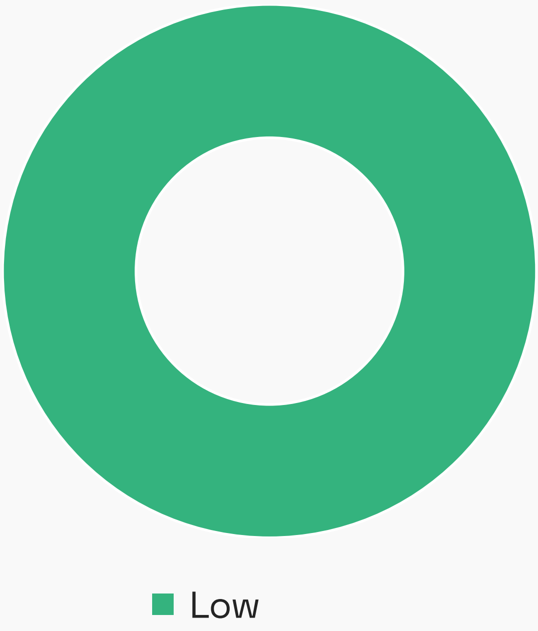
- ### Issue Symbolic Codes

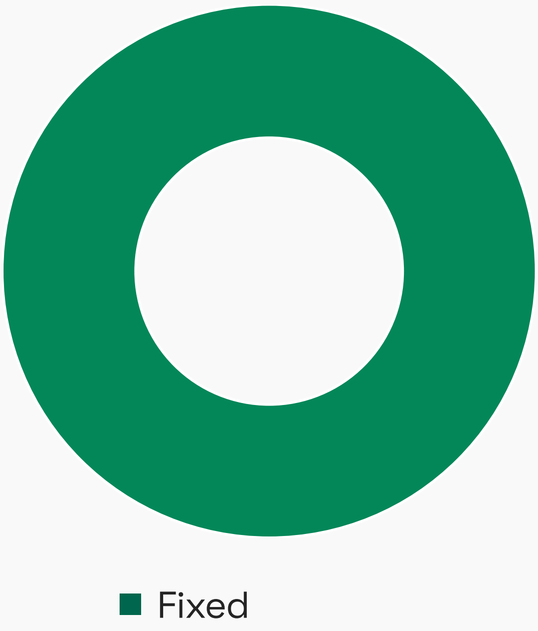
Each identified and validated issue is assigned a unique symbolic code during the security research stage.

Due to the structure of the vulnerability reporting flow, some rejected issues may be missing.

4. Findings Summary

Severity	Number of findings
🟥 Critical	0
🟥 High	0
🟧 Medium	0
🟩 Low	2
🟦 Informational	0
Total:	**2**





5. Weaknesses

This section contains the list of discovered weaknesses.

CITR-1 | Missing constraints on id_bn254_fr_b2n to be a valid BN254 number

Fixed ✓

Severity:	Low	Probability:	Unlikely	Impact:	Low

Path:

groth16_proof/circuits/verify_for_guest.circom

Description:

VerifyForGuest component has a **id_bn254_fr_bits[256]** input signal which values are assigned to the **id_bn254_fr_b2n** component and the result is constrained to be equal to **stark_verifier.codeRoot**.

```
// This gives the code root of the STARK circuit. Bytes of id_bn254_fr_bits reversed.
component id_bn254_fr_b2n = Bits2Num(256);
for (var i = 0; i < 32; i++) {
    for (var j = 0; j < 8; j++) {
        id_bn254_fr_b2n.in[255 - (8 * i + j)] <== id_bn254_fr_bits[8 * (31 - i) + j];
    }
}
...
stark_verifier.codeRoot === id_bn254_fr_b2n.out;
```

As the BN254 scalar field is a 254 bit number, **id_bn254_fr_b2n** signal being 256 bit array can contain an alias of a genuine 254 bit number.

Remediation:

Use **Bits2Num_strict** for **id_bn254_fr_b2n** and make **id_bn254_fr_bits** 254 signal array instead of 256.

CITR-2 | Missing binary constraints on VerifyForGuest circuit inputs

Fixed ✓

Severity:	Low	Probability:	Rare	Impact:	Low

Path:

groth16_proof/circuits/verify_for_guest.circom

Description:

The **VerifyForGuest** template contains Stark2Snark proof, Journal input signals and the Blake3 hash as an output signal.

```
template VerifyForGuest() {
    signal input iop[25749];
    signal input journal_digest_bits[256];
    signal input control_root[2];
    signal input pre_state_digest_bits[256];
    signal input post_state_digest_bits[256];
    signal input id_bn254_fr_bits[256];
    signal output final_blake3_digest;
    ...
}
```

The **journal_digest_bits**, **pre_state_digest_bits**, **post_state_digest_bits**, **id_bn254_fr_bits** signals are binary data, but the circuits are missing binary constraints on those signals which can cause to unexpected or invalid behavior during proof generation or verification. Without explicit binary constraints, these signals may take on invalid non-binary values that still satisfy other constraints in the circuit.

Remediation:

Add binary constraint on **journal_digest_bits**, **pre_state_digest_bits**, **post_state_digest_bits**, **id_bn254_fr_bits**.

hexens x Citrea