N1xyz nord
Security Review

Cantina Managed review by:

Mario Poneder, Security Researcher
Jerome Rousselot, Security Researcher

March 11, 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 High Risk . . . 4
  - 3.1.1 Withdrawal roots can be specifically crafted such that multiple withdrawals are possible 4
  - 3.1.2 Block finalization is immediate and permissionless . . . 5
- 3.2 Low Risk . . . 5
  - 3.2.1 Permissionless initialization allows anyone to set `operator` . . . 5
  - 3.2.2 Deposits can be spammed with zero amounts . . . 6
  - 3.2.3 Insufficient validation of tree height when computing subtree offset . . . 6
- 3.3 Informational . . . 6
  - 3.3.1 Project relies on vulnerable crate dependencies . . . 6

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

Null studios is a research-first infrastructure company building stronger, faster, and more resilient scaling systems for today's public and private blockchains.

From Mar 4th to Mar 8th the Cantina team conducted a review of n1xyz-nord on commit hash f072076a. The team identified a total of **6** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	2	2	0
Medium Risk	0	0	0
Low Risk	3	3	0
Gas Optimizations	0	0	0
Informational	1	0	1
Total	**6**	**5**	**1**

3 Findings

3.1 High Risk

3.1.1 Withdrawal roots can be specifically crafted such that multiple withdrawals are possible

Severity: High Risk

Context: withdraw.rs#L23-L30, withdraw.rs#L46-L67, merkle.rs#L40-L66

Description: The Merkle `root_from_proof` function below, will successfully skip the for-loop's body in case `leaves_len == 1` and an empty `proof`, i.e. `proof.len() == 0`.

```
pub fn root_from_proof<L>(
    leaf: L,
    leaf_idx: LeafIdx,
    leaves_len: LeafIdx,
    proof: &[[u8; 32]],
    digest_cb: impl Fn(L, &mut sha2::Sha256),
) -> [u8; 32] {
    let ceil_log2_n: u8 = leaves_len.checked_next_power_of_two().unwrap().ilog2() as u8;

    let mut root = hash_leaf(leaf, &digest_cb);
    let mut i: Height = 0;
    for h in 0..ceil_log2_n {    // @audit skipped if leaves_len == 1
        let begin = subtree_ofs(leaf_idx, h);
        if begin >= leaves_len {
            continue;
        }
        let is_leaf_left = leaf_idx & (1 << h) == 0;
        root = if is_leaf_left {
            hash_pair(root, proof[i as usize])
        } else {
            hash_pair(proof[i as usize], root)
        };
        i += 1;
    }
    assert_eq!(i as usize, proof.len());    // @audit passes if proof.len() == 0
    root    // @audit returns hash_leaf(leaf, &digest_cb) if loop is skipped
}
```

In this case, the `leaf_idx` parameter is ignored and the return value is just the digest of the `leaf`.

Attack vector: The operator who proposes a block can arbitrarily set the `withdrawal_root` allowing it to be just the digest of the `leaf_to_prove`. In this case, it suffices to invoke the `withdraw` instruction below with `claim.leaves_count == 1` and empty `claim.proof` to pass the root check and proceed with the withdrawal.

```
pub fn withdraw(ctx: Context<Withdraw>, claim: WithdrawalClaim) -> Result<()> {
    let root = ctx.accounts.state_update.facts.withdrawal_root;

    let leaf_to_prove = TransferParams {
        user: claim.user,
        mint: ctx.accounts.to_account.mint,
        amount: claim.amount,
    };
    let computed_root = crate::merkle::root_from_proof(
        &leaf_to_prove,
        claim.leaf_index,
        claim.leaves_count,
        &claim.proof,
        TransferParams::digest,
    );
    if root != computed_root {    // @audit will pass if withdrawal_root is digest of leaf_to_prove
        return err!(BridgeError::InvalidWithdrawalProof);
    }

    // ...
}
```

In order to avoid multiple withdrawals reusing the same block, the `Withdraw` context implements the `withdrawal_nullifier` below.

```
**[derive(Accounts)]:**

#[instruction(claim: WithdrawalClaim)]
pub struct Withdraw<'info> {
    // ...

    #[account(
        init,
        payer = payer,
        space = 8,
        seeds = [WITHDRAWAL_NULLIFIER_SEED, &claim.block_id.to_le_bytes(), &claim.leaf_index.to_le_bytes()],
        bump,
    )]
    pub withdrawal_nullifier: Account<'info, WithdrawalNullifier>,

    // ...
}
```

However, in case of an empty proof as discussed above, the `claim.leaf_index` is ignored in the root computation and therefore can be set to arbitrary values to bypass the `withdrawal_nullifier` allowing multiple withdrawals.

Recommendation: It is recommended to incorporate the `leaf_idx` into the root computation in any case and assert that `leaves_len > leaf idx` to also evaluate the `leaf_idx` in case of an empty `proof`.

Layer N: We now commit to the leaf index as of commit 7dba5a0a.

Cantina Managed: Fix verified.

3.1.2 Block finalization is immediate and permissionless

Severity: High Risk

Context: finalize_block.rs#L4-L37

Description: Once a block is proposed by the operator, which might malicious e.g. containing a specifically crafted withdrawal root, anyone can immediately invoke the `finalize_block` instruction to finalize it. Consequently, a malicious withdrawal can be performed immediately after proposing such a block.

Recommendation: It is recommended to introduce a challenge as well as a validity-proof mechanism that allows validators to challenge a block and prove its validity before it can be finalized.

Layer N: Fixed in commit d74b33e9.

Cantina Managed: Fix verified. For the testnet deployment the operator is trusted, but the team will introduce a minimum delay before finalization.

3.2 Low Risk

3.2.1 Permissionless initialization allows anyone to set `operator`

Severity: Low Risk

Context: initialize.rs#L5-L43

Description: The `initialize` instruction and its account context impose no restrictions on the signer. Consequently, the first one to invoke `initialize` after deployment is in charge of setting the protocol's `operator` role.

Recommendation: It is recommended to add the following code to the `Initialize` context to require co-signing of the `initialize` instruction with the program's private key (keypair) that should only be known to the deployer.

```
**[account(address = crate::ID)]:** pub program: Signer<'info>
```

Layer N: Fixed in commit cc527d17.

Cantina Managed: Fix verified.

3.2.2 Deposits can be spammed with zero amounts

Severity: Low Risk

Context: deposit_spl.rs#L55-L95

Description: The permissionless `deposit_spl` instruction imposes no restrictions on the `amount`. Consequently, anyone can spam deposits with zero or dust amounts thereby unnecessarily increasing the `last_deposit_index`.

Recommendation: It is recommended to introduce a minimum deposit amount.

Layer N: Fixed in commit 693853f0.

Cantina Managed: Fix verified. The operator could define a minimum deposit amount when whitelisting an asset. This amount is asset-dependent, as each asset's financial value is unique. An instruction to update whitelisted asset minimum deposit amounts would be useful in case of errors during whitelisting or when asset valuations change significantly. In the long term, integrating with a DEX could help decentralise this process.

3.2.3 Insufficient validation of tree height when computing subtree offset

Severity: Low Risk

Context: merkle.rs#L116-L125

Description: The following shift operation in the `subtree_ofs` function can be performed with any given tree height `h`.

```
let mask0: LeafIdx = 1 << h;
```

However, the `LeafIdx` type is defined as `u64`. Consequently, any tree height `h >= 64` leads to invalid results.

Recommendation: It is recommended to assert that `h < 64`.

Layer N: Fixed in commit 7dba5a0a.

Cantina Managed: Fix verified.

3.3 Informational

3.3.1 Project relies on vulnerable crate dependencies

Severity: Informational

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

Description: The following vulnerable crate dependencies were identified using `cargo audit`:

1. Timing variability in `curve25519-dalek`'s `Scalar29::sub`/`Scalar52::sub`:

```
Crate:     curve25519-dalek
Version:   3.2.0
Title:     Timing variability in `curve25519-dalek`'s `Scalar29::sub`/`Scalar52::sub`
Date:      2024-06-18
ID:        RUSTSEC-2024-0344
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0344
Solution:  Upgrade to >=4.1.3
```

2. Double Public Key Signing Function Oracle Attack on `ed25519-dalek`:

```
Crate:     ed25519-dalek
Version:   1.0.1
Title:     Double Public Key Signing Function Oracle Attack on `ed25519-dalek`
Date:      2022-06-11
ID:        RUSTSEC-2022-0093
URL:       https://rustsec.org/advisories/RUSTSEC-2022-0093
Solution:  Upgrade to >=2
```

3. `ssl::select_next_proto` use after free:

```
Crate:     openssl
Version:   0.10.68
Title:     `ssl::select_next_proto` use after free
Date:      2025-02-02
ID:        RUSTSEC-2025-0004
URL:       https://rustsec.org/advisories/RUSTSEC-2025-0004
Solution:  Upgrade to >=0.10.70
```

4. Marvin Attack: potential key recovery through timing sidechannels:

```
Crate:     rsa
Version:   0.9.7
Title:     Marvin Attack: potential key recovery through timing sidechannels
Date:      2023-11-22
ID:        RUSTSEC-2023-0071
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0071
Severity:  5.9 (medium)
Solution:  No fixed upgrade is available!
```

Recommendation: It is recommended to upgrade the affected crate dependencies if applicable and viable in terms of compatibility.

Layer N: Acknowledged.

Cantina Managed: Acknowledged.