CANTINA

N1xyz Nord Proton

Security Review

Cantina Managed review by:

M4rio, Lead Security Researcher

Frankcastle, Security Researcher

0xLuk3, Associate Security Researcher

August 15, 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

Layer N is Ethereum at internet-scale: the first highly scalable blockchain capable of enabling up to hundreds of thousands of transactions per second, sub-ms latencies, and dedicated congestion-free compute for applications.

From Jul 1st to Aug 15th the Cantina team conducted a review of n1xyz-proton on commit hash b5f95c13. The team identified a total of **11** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	1	1	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	4	4	0
Gas Optimizations	0	0	0
Informational	5	4	1
Total	**11**	**10**	**1**

3 Findings

3.1 Critical Risk

3.1.1 Permanent DoS via funding un-initialised PDA accounts

Severity: Critical Risk

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

Finding Description: Because anyone can transfer lamports to any address, an attacker can pre-fund the deterministic PDA addresses that the bridge will later create. Several PDAs are instantiated with `system_instruction::create_account`:

- `challenge_block` - challenge-nullifier PDA.
- `withdraw` - effect-nullifier PDA.
- `set_permission` - ACL-entry PDA.

`create_account` (see lines 160-168 of the System Program) fails if the target account already holds lamports. In the surrounding logic of the aforementioned PDAs, the program also checks whether the PDA is already created; if it holds lamports it verifies the owner and reverts when the owner is not the bridge program, e.g.:

- `challenge_block.rs`:

```
#[account(mut, seeds = [
    CHALLENGE_NULLIFIER_SEED,
    &bridge.key().to_bytes(),
    &block_id.to_le_bytes(),
    &validator.key().to_bytes(),
], bump)]
pub challenge_nullifier: AccountInfo<'info>,

// .....

pub fn challenge_block(ctx: Context<ChallengeBlock>, block_id: u64) -> Result<()> {
// ...
    if **ctx.accounts.challenge_nullifier.lamports.borrow() > 0 {
        assert_eq!(*ctx.accounts.challenge_nullifier.owner, ctx.accounts.program.key());
        return err!(crate::BridgeError::BlockAlreadyChallenged);
    }
// ...
}
```

By pre-funding these PDAs with a minimal lamport balance, an attacker can permanently block their creation and thus DoS the associated functionality. The most critical impact is on `challenge_block`, where a validator could be prevented from challenging a malicious block.

Recommendation: Consider doing what Anchor is doing when it tries to create an account and detects that the account already has lamports:

- Allocate.
- Assign.
- Make sure it's rent extempt.

constraints.rs#L1648-L1675

The following cases should be handled:

- If the lamports, the data field (discriminator), and the account ownership all match, and the account has valid data, we should revert with an "already initialized" error.
- If only lamports exist (lamports > `0`), but the account is still owned by the system program or the data field is empty, we should perform manual account creation.
- If no lamports exist (lamports == `0`) and the account is owned by the system program, this means the account has no data-so using the system instruction create_account is valid here.

Layer N: Fixed in commit 34dcb95d.

Cantina Managed: Fix verified. It would be beneficial for the project to add some testing for this, if possible, at least unit tests.

3.2 Medium Risk

3.2.1 Unfinalized Blocks Cannot Be Pruned Due to Effects Execution Check

Severity: Medium Risk

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

Description: The current logic prevents pruning of unfinalized but expired blocks due to a strict check that compares `block.facts.effects_count` with `block.effects_executed`. This check assumes all block effects must have been executed, which is only valid for finalized blocks.

However, approved but unfinalized blocks-especially those that are expired or challenged-may not have had their effects executed. Enforcing this check on such blocks causes a permanent revert when trying to prune them. As a result:

- The refund account is never refunded.
- The block accounts remain open indefinitely.

Additionally, for challenged blocks (marked invalid by validators), it's expected that their effects should not be executed-yet this check still applies, incorrectly halting cleanup. Here's the problematic condition:

```
require!(
    block.facts.effects_count == block.effects_executed,
    BridgeError::BlockHasUnexecutedEffects
);
```

This will always fail for unfinalized blocks, since `block.effects_executed` remains `0`, while `block.facts.effects_count` is non-zero.

Recommendation: Restrict the execution check to finalized blocks only, where we are certain the effects must have been executed. Suggested logic update:

```
let is_finalizable = IsChallengePeriodExpired {
    slot_current: Clock::get().unwrap().slot,
    slot_proposed: block.slot_proposed,
    slots_challenge_period: ctx.accounts.bridge.challenge_period_slots,
}.run() && block.challenges < ctx.accounts.bridge.challenge_consensus_threshold;

if block.approval == Approval::Finalized || is_finalizable {
    assert!(block.facts.effects_count >= block.effects_executed);
    require!(
        block.facts.effects_count == block.effects_executed,
        BridgeError::BlockHasUnexecutedEffects
    );
}
```

This ensures:

- Finalized blocks are held to strict effect execution guarantees.
- Invalid or unfinalizable blocks can still be safely pruned.

Layer N: Fixed in commit 7ea96be6.

Cantina Managed: The current implementation has a lifetime check only if a block is not finalized. If a block is the second to last finalized, it can be pruned immediately without lifetime check. Also the current implementation let's the last block (if it's not finalized) to be pruned if the lifetime check passes e.g.

- Propose block B1.
- B1 was challenged
- No new blocks where proposed
- Lifetime has passed from the time B1 was proposed.

B1 can be pruned even if currently it's the last block that was proposed.

Layer N: This is by design and we acknowledge these properties of this design.

3.3 Low Risk

3.3.1 Whitelisted Assets Cannot Be Disabled or Blacklisted After Initialization

Severity: Low Risk

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

Description: The current asset configuration system allows for whitelisting assets using a deterministic PDA derived from the bridge and mint addresses:

```
#[account(
    init,
    payer = payer,
    space = 8 + AssetConfig::INIT_SPACE,
    seeds = [
        ASSET_CONFIG_SEED,
        &bridge.key().to_bytes(),
        &mint.key().to_bytes()
    ],
    bump
)]
pub asset_config: Account<'info, AssetConfig>,
```

However, there is no mechanism to blacklist or disable an asset once it has been whitelisted. In other words, once an `asset_config` account is created, the protocol does not provide any built-in way to deactivate or restrict the asset's usage afterward. Attempts to prevent deposits by setting `min_deposit` to zero are also explicitly disallowed. This behavior is enforced in the `set_min_deposit` instruction. This means once an asset is added, it remains usable indefinitely-even if a critical issue is discovered or the asset becomes malicious or deprecated.

Recommendation: Introduce a mechanism to deactivate or blacklist assets after they have been whitelisted. This could be achieved by:

- Adding a boolean field like `is_enabled` or `is_blacklisted` in the `AssetConfig` struct.
- Updating all relevant deposit and transfer logic to respect this flag and revert if the asset is disabled.
- Optionally allowing setting `min_deposit = 0` as a soft-disable, if stricter blacklisting is not implemented.

Layer N: Fixed in commit 86e6a58b.

Cantina Managed: Fix verified.

3.3.2 Dummy Block Spam Can Force Premature Pruning of Legitimate Blocks

Severity: Low Risk

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

Description: The pruning mechanism currently determines a block's prunability by comparing its ID to the last proposed block ID. This opens the door for a subtle but impactful vulnerability: an operator can spam the system with many dummy or invalid block proposals, rapidly increasing the `last_block_id`.

Because pruning decisions are ID-based rather than time- or slot-based, unexpired yet valid blocks may be misclassified as prunable much earlier than intended. Although only blocks with executed effects or zero effects are immediately pruned (thanks to a safeguard), this behavior still puts legitimate blocks at risk of premature deletion in scenarios where effects are processed slowly or blocked. This manipulation could lead to:

- Loss of valid block data before its expected lifetime.
- Unintentional halting of cross-chain message execution.

Recommendation: To prevent this type of manipulation and ensure block lifetime is preserved as intended:

1. Use Slot-Based Expiry: Define block prunability based on elapsed slots since `slot_proposed`, rather than relative block ID.
2. Compare Against Last Finalized Block ID: Use the last finalized block ID instead of the last proposed block ID for pruning decisions. This avoids counting non-final, potentially malicious proposals toward pruning logic.

Layer N: Fixed in commit a7100f10. Switched to slot based `block_lifetime`.

Cantina Managed: Fix verified.

3.3.3 Incorrect freeze crumb reporting

Severity: Low Risk

Context: freeze.rs#L24

Finding Description: The `freeze` instruction correctly updates on-chain state (`ctx.accounts.bridge.frozen = freeze`), but the event emitted for off-chain observers is hard-coded:

```
crate::crumbs::Crumb::Freeze { freeze: true }
```

Regardless of whether the caller is freezing **or un-freezing**, every `Freeze` crumb reports `freeze: true`.

Recommendation: Emit the actual user-supplied value:

```
crate::crumbs::emit_cpi(
    &crate::crumbs::Crumb::Freeze { freeze }.versioned(),
    &ctx.accounts.program,
    &ctx.accounts.bridge,
    ctx.accounts.crumb_authority.to_account_info(),
    ctx.bumps.crumb_authority,
);
```

Layer N: Fixed in commit b4fe4289.

Cantina Managed: Fix verified.

3.3.4 Deposit inflation with Fee-On-Transfer tokens

Severity: Low Risk

Context: deposit_create.rs#L73-L92

Finding Description: `deposit_create` uses the deprecated unchecked SPL-Token `transfer`:

```
#[allow(deprecated)]
token_interface::transfer( ... authority: payer ..., amount )?;
```

For tokens that enforce a transfer fee or burn ("fee-on-transfer"), the receiving vault (`token_account`) ends up with less than `amount`. However, the program records the full `amount` inside the `UnqueuedDeposit`:

```
*cx.accounts.deposit = UnqueuedDeposit { transfer: TransferParams { ... amount } };
```

This will result in inflated deposits - on-chain state claims more tokens than were actually credited to the bridge vault.

Recommendation: Switch to `transfer_checked_with_fee` so the program respects the mint's decimals and fee behaviour of the fee. The team has acknowledged this limitation and, for the moment, whitelists only standard, zero-fee SPL tokens until full support for special behaviours is added.

Layer N: Fixed in commit 823e2da6. Updated the TODO with a note about using `transfer_checked_with_fee`.

Cantina Managed: Fix verified.

3.4 Informational

3.4.1 Unfinished developments / TODO comments

Severity: Informational

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

Description: The codebase contains multiple `TODO` comments indicating incomplete implementations and pending design decisions. Notable examples include unimplemented bridge transfer in bridge/src/lib.rs:103, unimplemented block approval in bridge/src/lib.rs:153, or use missing checked transfer implementation in bridge/src/instructions/withdraw.rs:162, other occurrences are present in:

- `merkle.rs:105`.
- `lib.rs:20`.
- `instructions/initialize_bridge.rs:10`.
- `instructions/xbridge_transfer.rs`, multiple occurrences.
- `instructions/deposit_create.rs`, multiple occurrences.
- `state/types.rs`, multiple occurrences.

These incomplete implementations reduce code maintainability and may lead to unexpected behavior when attempting to use partially implemented features.

Recommendation: Address each `TODO` systematically before final production-ready deployment.

Layer N: Acknowledged.

Cantina Managed: Acknowledged.

3.4.2 Typographical error in RPC interface structure

Severity: Informational

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

Description: The RPC module in rpc.rs contains a struct named `SetWinwodSizeIx` which appears to be a typographical error for `SetWindowSizeIx`. While this naming inconsistency doesn't affect runtime behavior, it may lead to confusion or issues when integrating with the RPC interface.

Recommendation: Rename `SetWinwodSizeIx` to `SetWindowSizeIx`.

Layer N: Fixed in commit 657c8dab.

Cantina Managed: Fix verified.

3.4.3 Missing parameter guards for `challenge_consensus_threshold` and `challenge_period_slots`

Severity: Informational

Context: initialize_bridge.rs#L63-L64

Finding Description: `SetChallengeConsensusThreshold` and `SetChallengePeriodSlots` allow the bridge owner to change:

- `challenge_consensus_threshold` - Number of validator challenges required to block finalization.
- `challenge_period_slots` - Duration (in slots) during which challenges are accepted.

Neither instruction validates the new value. As a result the owner can set:

Extreme value	Consequence
`challenge_consensus_threshold = 0`	*Any* block can be finalized im
`challenge_consensus_threshold > u16::MAX` (via future refactor) or validator set size	Every block becomes unfina
`challenge_period_slots = 0`	Challenge window closes in
`challenge_period_slots` extremely large (e.g. u64::MAX)	Blocks remain “Proposed” fo

Recommendation: We agree that these values can be changed freely in emergency circumstances, but we advise against changing them often - especially if doing so could allow the previous challenged blocks to be finalized.

Layer N: Fixed in commit 4646d536. Updated doc comment in fields with above.

Cantina Managed: Fix verified.

3.4.4 Centralize the challenge-period expiry check

Severity: Informational

Context: challenge_block.rs#L127-L144

Finding Description: The helper struct `IsChallengePeriodExpired` is defined directly inside `challenge_block.rs`. Other instructions like `finalize_block` rely on the *same* rule but must import or re-implement the logic manually.

Recommendation: Move `IsChallengePeriodExpired` into a shared `utils` module and re-export it.

Layer N: Fixed in commit fe077523.

Cantina Managed: Fix verified.

3.4.5 Orphaned DA-Fact accounts accumulate rent

Severity: Informational

Context: prune_block.rs#L33

Finding Description: Each call to `finalize_da_fact` creates a dedicated PDA (`FactStateStorage`) for a single DA-fact and marks it `Finalized`. During `propose_block`, the block merely checks that the referenced DA-fact is already `Finalized`; after the block is itself finalised and eventually pruned, the DA-fact account is never touched again. Because there is no corresponding "prune-da-fact" logic, all fact-storage PDAs persist indefinitely, even though each fact is consumed by exactly one block.

Recommendation: Add a lightweight `prune_da_fact` instruction that can be called permissionlessly once the block that consumed the fact is prunable (e.g., `block_id < last_block_id - block_lifetime`). Furthermore, consider analyzing other accounts that might need to be pruned as well once the block is pruned.

Layer N: Fixed in PR 1621, removed all the dummy DA logic from the bridge on master.

Cantina Managed: Fix verified.