Burrow Finance

Burrowland - Smart Contracts Update

Security Assessment Report

Version: 2.0

May, 2025

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Burrow Finance components. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the components in scope. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Burrow Finance components contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Burrow Finance components in scope.

Overview

Burrowland is a decentralised lending platform built on the NEAR blockchain. It enables users to lend and borrow assets while earning interest on their deposits.

This review primarily focused on recent protocol improvements. Key updates include restricted owner management, improved liquidation mechanisms including direct margin trading position liquidation, margin base token limits, protocol debt tracking, and safety measures for incomplete position actions.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the burrowland repository.

The scope of this time-boxed review was strictly limited to files at commit 44ea182.

Note: third party libraries and dependencies were excluded from the scope of this assessment.

Approach

The security assessment covered components written in Rust.

For the Rust components, the manual review focused on identifying issues associated with the business logic implementation of the components in scope. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Rust language.

Additionally, the manual review process focused on identifying vulnerabilities related to known Rust anti-patterns and attack vectors, such as unsafe code blocks, integer overflow, floating point underflow, deadlocking, error handling, memory and CPU exhaustion attacks, and various panic scenarios including index out of bounds, panic!(), unwrap(), and unreachable!() calls.

To support the Rust components of the review, the testing team also utilised the following automated testing tools:

- Clippy linting: `https://doc.rust-lang.org/stable/clippy/index.html`
- Cargo Audit: `https://github.com/RustSec/rustsec/tree/main/cargo-audit`
- Cargo Outdated: `https://github.com/kbknapp/cargo-outdated`
- Cargo Geiger: `https://github.com/rust-secure-code/cargo-geiger`
- Cargo Tarpaulin: `https://crates.io/crates/cargo-tarpaulin`

Output for these automated tools is available upon request.

Coverage Limitations

Due to the time-boxed nature of this review, all documented vulnerabilities reflect best effort within the allotted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 6 issues during this assessment. Categorised by their severity:

- Low: 1 issue.
- Informational: 5 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Burrow Finance components in scope. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the components, including optimisations, are also described in this section and are labelled as "informational".

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
BUR2-01	Vulnerable Dependencies	Low	Closed
BUR2-02	Missing Liquidation Rate Lower Bounds	Informational	Closed
BUR2-03	Unbounded Default Margin Base Token Limits	Informational	Closed
BUR2-04	Invalid AccountId Creation In Farm ID Generation	Informational	Closed
BUR2-05	Position Removed Before Liquidation Validation	Informational	Closed
BUR2-06	Miscellaneous General Comments	Informational	Resolved

BUR2-01	Vulnerable Dependencies		
Asset	All Files		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

A number of dependencies used in the project are outdated and potentially vulnerable. The following dependencies are outdated:

- `curve25519-dalek` at `3.2.0`
- `ed25519-dalek` at `1.0.1`
- `hashbrown` at `0.15.0`
- `idna` at `0.4.0`
- `mio` at `0.8.9`
- `openssl` at `0.10.59`
- `ring` at `0.17.8`
- `rustls` at `0.23.13`

The issue has been rated as low likelihood and low impact, as the vulnerable dependencies are not directly related to the core functionality of the smart contracts. However, it is still important to keep dependencies up to date to ensure the security and stability of the codebase.

Recommendations

It is recommended to add `cargo audit` to the CI process.

Where possible update all dependencies to the latest version. Additionally, ensure each of the following direct and indirect dependencies are updated as follows:

- `curve25519-dalek` to >=4.1.3
- `ed25519-dalek` to >=2
- `hashbrown` to >=0.15.1
- `idna` to >=1.0.0
- `mio` to >=0.8.11
- `openssl` to >=0.10.72
- `ring` to >=0.17.12
- `rustls` to >=0.23.18

Resolution

The development team has acknowledged the comments and suggestions and will monitor dependencies for updates.

BUR2-02	Missing Liquidation Rate Lower Bounds
Asset	`margin_base_token_limit.rs`
Status	**Closed:** See Resolution
Rating	Informational

Description

The `set_margin_base_token_limit()` function validates that the combined liquidation benefit rates do not exceed the maximum ratio, but there is no minimum requirement for these rates. This could lead to insufficient liquidation incentives.

```
require!(
    mbtl.liq_benefit_protocol_rate + mbtl.liq_benefit_liquidator_rate <= MAX_RATIO,
    "require: liq_benefit_protocol_rate + liq_benefit_liquidator_rate <= MAX_RATIO"
);
```

Without a lower bound on these rates, administrators could set them to zero or very low values.

Recommendations

Implement a minimum threshold for the combined liquidation benefit rates to ensure there are always sufficient incentives for liquidators.

Resolution

The development team are aware of the edge case, this is a design decision which allows for flexibility in setting liquidation rates.

BUR2-03	Unbounded Default Margin Base Token Limits
Asset	`margin_base_token_limit.rs`
Status	**Closed:** See Resolution
Rating	Informational

Description

The `default_limit()` function in `MarginBaseTokenLimit` sets extremely permissive default values. The critical parameters controlling position sizes and total token availability are set to the maximum possible value `(u128::MAX)`.

```
pub fn default_limit(margin_config: &MarginConfig) -> Self {
    Self {
        max_base_token_short_position: u128::MAX,
        max_base_token_long_position: u128::MAX,
        total_base_token_available_short: u128::MAX,
        total_base_token_available_long: u128::MAX,
        // other parameters...
    }
}
```

If these default values are used without explicit override, they effectively remove any cap on position sizes or total token availability.

Recommendations

Set reasonable, conservative default values for position size and token availability parameters that limit risk exposure until administrators explicitly configure appropriate values. Also consider rejecting any token configuration that uses the default permissive values.

Resolution

The issue will not be patched in the code as the administration operation flow is to set these limits first before adding base tokens.

BUR2-04	Invalid AccountId Creation In Farm ID Generation
Asset	account.rs
Status	**Closed:** See Resolution
Rating	Informational

Description

The `get_all_potential_farms()` function creates `AccountIds` from token strings using `new_unchecked()`, bypassing NEAR's account ID validation. While this does not present an immediate security risk, it could lead to unexpected behaviour if invalid account IDs are processed.

When processing LP token positions, the function converts position strings directly into `AccountIds`:

```
Position::LPTokenPosition(lp_token_position) => {
    potential_farms.insert(FarmId::Supplied(AccountId::new_unchecked(position.clone())));  // @audit No validation of AccountId
          ↪  format
    potential_farms.insert(FarmId::TokenNetBalance(AccountId::new_unchecked(position.clone())));
    // ...
}
```

Using `new_unchecked()` skips the validation that ensures the `AccountId` follows NEAR's naming rules.

Recommendations

Replace `new_unchecked()` with proper validation using `try_from()` for better security practices.

Resolution

As the current code path validates the account ID format before calling `new_unchecked()`, the development team has decided to keep the current implementation.

BUR2-05	Position Removed Before Liquidation Validation
Asset	`margin_position.rs`
Status	**Closed:** See Resolution
Rating	Informational

Description

The `process_margin_liquidate_direct()` function removes a margin trading position from storage before validating that it is actually eligible for liquidation, potentially creating a race condition in case of validation failure.

In the current implementation, the function first removes the position from the owner's account using `margin_positions.remove(pos_id)` and only after checks if the position is liquidatable with `is_mt_liquidatable()`. While NEAR's transaction atomicity should roll back all state changes if the transaction fails, this approach is error-prone and could lead to inconsistent state if there are bugs in error handling or partial execution scenarios.

Recommendations

Restructure the function to validate all liquidation conditions before making any state changes. First retrieve the position without removing it, check all conditions, and only then proceed with removing the position and executing the liquidation.

Resolution

The development team has acknowledged the and intend to keep the current implementation, to keep with similar processing patterns in the ecosystem.

BUR2-06	Miscellaneous General Comments
Asset	All contracts
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Missing Event Emission For Margin Base Token Limit Removal**

   Related Asset(s): margin_base_token_limit.rs

   The `remove_margin_base_token_limit()` function removes a margin base token limit but does not emit an event to record this critical action. This creates a gap in the protocol's audit trail and reduces transparency for users and monitoring systems.

   ```
   #[payable]
   pub fn remove_margin_base_token_limit(&mut self, token_id: TokenId) {
       self.assert_owner();
       self.margin_base_token_limits.remove(&token_id);
   }
   ```

   While this does not directly impact the security of the protocol, it reduces visibility into administrative actions that affect risk parameters, making it harder to track changes and potentially delaying the detection of unauthorised modifications.

   Emit an event when removing margin base token limits to ensure all critical administrative actions are properly logged.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team has acknowledged the comments and suggestions and will consider implementing them in future releases.

Appendix A Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

σ'