# Security Audit Report for Burrow

**Date:** Mar 19th, 2022

**Version:** 2.0

**Contact**: contact@blocksecteam.com

BlockSec
Audit Team

Digitally signed by
BlockSec Audit Team
Date: 2022.03.22
11:31:11 +08'00'

# Contents

## Report Manifest

| Item | Description |
|---|---|
| Client | Burrow |
| Target | Burrow |

## Version History

| Version | Date | Description |
|---|---|---|
| 1.0 | Feb 14th, 2022 | First Release |
| 2.0 | Mar 19th, 2022 | Second Release |

**About BlockSec** The BlockSec Team focuses on the security of the blockchain ecosystem, and collaborates with leading DeFi projects to secure their products. The team is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and released detailed analysis reports of high-impact security incidents. They can be reached at Email, Twitter and Medium.

# Chapter 1 Introduction

## 1.1 About Target Contracts

| Information | Description |
|---|---|
| Type | Smart Contract |
| Language | Rust |
| Approach | Semi-automatic and manual verification |

The audit scope includes the contract under the directory contract in the burrowland repository [1].

The auditing process is iterative. Specifically, we will further audit the commits that fix the founding issues. If there are new issues, we will continue this process. Thus, there are multiple commit SHA values referred in this report. The commit SHA values before and after the audit are shown in the following.

**Before and during the audit**

| Contract Name | Stage | Commit SHA |
|---|---|---|
| Burrow | Initial | 0de39e9876694b0926c557426e7608b96444ee8e |
| Burrow | Version 1.0 | 3ff104ca59fc22cc2ab64fc43b38e929443ced2f |

**After**

| Project | Commit SHA |
|---|---|
| Burrow | 6fe609bd53d170508366f70eb5d00a7574e2b94b |

## 1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

[1] https://github.com/NearDeFi/burrowland

## 1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

### 1.3.1 Software Security

- Reentrancy
- DoS
- Access control
- Data handling and data flow
- Exception handling
- Untrusted external call and control flow
- Initialization consistency
- Events operation
- Error-prone randomness
- Improper use of the proxy system

### 1.3.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Access control
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

### 1.3.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

### 1.3.4 Additional Recommendation

- Gas optimization
- Code quality and style

**Note** *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

## 1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. Accordingly, the severity measured in this report are classified into four categories: **High**, **Medium**, **Low** and **Undetermined**.

Furthermore, the status of a discovered issue will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The issue has been received by the client, but not confirmed yet.
- **Confirmed** The issue has been recognized by the client, but not fixed yet.
- **Fixed** The issue has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

# Chapter 2 Findings

In total, we find 7 potential issues in the smart contract. We also have 8 recommendations, as follows:

- High Risk: 0
- Medium Risk: 5
- Low Risk: 2
- Recommendations: 8

The details are provided in the following sections.

| ID | Severity | Description | Category | Status |
|---|---|---|---|---|
| 1 | Medium | *MAX_NUM_ASSETS is a constant* | Software Security | Fixed |
| 2 | Medium | *Missing check on the returned value* | Software Security | Fixed |
| 3 | Medium | *Storage usage is wrong in liquidation action* | DeFi Security | Fixed |
| 4 | Medium | *Storage usage is wrong in setting users' storage* | DeFi Security | Fixed |
| 5 | Low | *Incomplete asset configuration check* | DeFi Security | Fixed |
| 6 | Medium | *The freshness of PriceData should be checked* | DeFi Security | Fixed |
| 7 | Low | *Missing check in init function* | Defi Security | Fixed |
| 8 | - | *Dead code is found* | Recommendation | Fixed |
| 9 | - | *Deflation Token is not supported* | Recommendation | Confirmed |
| 10 | - | *Redundant invoked functions* | Recommendation | Fixed |
| 11 | - | *Reliability of Oracle* | Recommendation | Confirmed |
| 12 | - | *Upgrade module of the contract* | Recommendation | Fixed |
| 13 | - | *The risk of centralized design* | Recommendation | Confirmed |
| 14 | - | *Allowed actions* | Recommendation | Acknowledged |
| 15 | - | *Conflicts in Documentation* | Recommendation | Fixed |

## 2.1 Software Security

### 2.1.1 MAX_NUM_ASSETS cannot be changed without upgrading the contract

**Status** Fixed.

**Description** This issue is introduced in or before the initial commit. The MAX_NUM_ASSETS in line 135 are used in assert statement. However, MAX_NUM_ASSETS is a constant (i.e., 10). In this case, the assert statement in line 135 may fail when new asset is added. Part of the functionality cannot be guaranteed if the assert fail.

```
134 if need_number_check {
135   assert!(account.collateral.len() + account.borrowed.len() <= MAX_NUM_ASSETS);
136 }
```

**Listing 2.1:** contract/src/action.rs

**Impact** Part of the functionality may not work after adding new assets.

**Suggestion I** Add an interface to set the MAX_NUM_ASSETS.

### 2.1.2 Missing check on the returned value

**Status** Fixed.

**Description** This issue is introduced in or before the initial commit. In function shares_to_amount, when the shares.0 < self.balance and shares.0 > self.shares.0, the calculated return value will be larger than the self.balance.

```
pub fn shares_to_amount(&self, shares: Shares, round_up: bool) -> Balance {
  if shares.0 >= self.balance || shares.0 == self.shares.0 {
    self.balance
  } else {
    let extra = if round_up {
      U256::from(self.shares.0 - 1)
    } else {
      U256::zero()
    };
    ((U256::from(self.balance) * U256::from(shares.0) + extra) / U256::from(self.shares.0))
    .as_u128()
  }
}
```

**Listing 2.2:** contract/src/pool.rs

**Impact** The return value of function shares_to_amount may be larger than the self.balance.

**Suggestion I** Check whether the return value is larger than the self.balance.

## 2.2 DeFi Security

### 2.2.1 Storage usage is wrong in liquidation action

**Status** Fixed.

**Description** This issue is introduced in or before the initial commit. The `required_bytes` in line 401 includes all the other accounts' storage usages. In this case, comparing between `available_bytes` and `required_bytes` does not make any sense. The `available_bytes` are for liquidation account and should not be added into the account who execute the liquidation action. `Extra_bytes` should not be added into the `liquidation_account` in line 409, the logic here is wrong.

```
394// We have to adjust the initial_storage_usage for the acting account to not double count
395// the bytes from the liquidation account. As well as potentially cover the extra bytes
396// required by the liquidation account that might be added due to farms.
397let current_storage_usage = env::storage_usage();
398if current_storage_usage > liquidation_storage.initial_storage_usage {
399  let required_bytes = current_storage_usage - liquidation_storage.initial_storage_usage;
400  let available_bytes = liquidation_storage.available_bytes();
401  if available_bytes < required_bytes {
402    let extra_bytes = required_bytes - available_bytes;
403    log!(
404    "Account {} has to cover extra storage of {} bytes for liquidation account {}",
405    account_id,
406    extra_bytes,
407    liquidation_account_id,
408    );
409    liquidation_storage.initial_storage_usage += extra_bytes;
410    storage.initial_storage_usage += available_bytes;
411  } else {
412    storage.initial_storage_usage += required_bytes;
413  }
414} else {
415  let released_bytes = liquidation_storage.initial_storage_usage - current_storage_usage;
416  storage.initial_storage_usage -= released_bytes;
417}
```

**Listing 2.3:** contract/src/action.rs

**Impact** The total calculation of the storage usage during liquidation process is wrong, which can influence the storage usage of users.

**Suggestion I** Re-implement the whole logic.

### 2.2.2 Storage usage is wrong in setting users' storage

**Status** Fixed.

**Description** This issue is introduced in or before the initial commit. The `env::storage_usage()` returns the storage usage of the whole contract while the `storage.initial_storage_usage` in line 78 is for one account. Comparing these two values does not make sense. The calculated `extra_bytes_used` in line 79 includes all the other accounts' used storage and should not be added into the `storage.used_bytes` in line 80. Here the logic is wrong.

```
76pub fn internal_set_storage(&mut self, account_id: &AccountId, mut storage: Storage) {
77  let storage_usage = env::storage_usage();
78  if storage_usage > storage.initial_storage_usage {
79    let extra_bytes_used = storage_usage - storage.initial_storage_usage;
80    storage.used_bytes += extra_bytes_used;
```

```
    storage.assert_storage_covered();
  } else {
    let bytes_released = storage.initial_storage_usage - storage_usage;
    assert!(
    storage.used_bytes >= bytes_released,
    "Internal storage accounting bug"
    );
    storage.used_bytes -= bytes_released;
  }
  self.storage.insert(account_id, &storage.into());
}
```

Listing 2.4: contract/src/actions.rs

**Impact** The logic of setting the storage usage for each account is wrong, which can influence the storage usage of users.

**Suggestion I** Re-implement the whole logic.

### 2.2.3 Incomplete asset configuration check

**Status** Fixed.

**Description** This issue is introduced in or before the initial commit. In function `update_asset`, `assert_valid` is invoked to check whether the updated configuration is valid.

```
#[payable]
pub fn update_asset(&mut self, token_id: ValidAccountId, asset_config: AssetConfig) {
  assert_one_yocto();
  asset_config.assert_valid();
  self.assert_owner();
  let mut asset = self.internal_unwrap_asset(token_id.as_ref());
  asset.config = asset_config;
  self.internal_set_asset(token_id.as_ref(), asset);
  }
```

Listing 2.5: contract/src/config.rs

However, the validation check is incomplete. For example, the `extra_decimal` should be the same as the original one and cannot be changed. Meanwhile, the `volatility_ratio` should be within 0-10000.

```
pub fn assert_valid(&self) {
  assert!(self.reserve_ratio <= MAX_RATIO);
  assert!(self.target_utilization < MAX_POS);
  assert!(self.target_utilization_rate.0 <= self.max_utilization_rate.0);
}
```

Listing 2.6: contract/src/asset_config.rs

**Impact** The updated `asset_config` may not be valid.

**Suggestion I** Add the check for the attribute `extra_decimal` and `volatility_ratio`.

### 2.2.4 The freshness of PriceData should be checked

**Status** Fixed.

**Description** This issue is introduced in or before the initial commit. `PriceData.timestamp` and `PriceData.recency_duration_sec` are unchecked when `PriceData` is converted into `Prices`.

```rust
pub struct PriceData {
  #[serde(with = "u64_dec_format")]
  pub timestamp: Timestamp,
  pub recency_duration_sec: DurationSec,

  pub prices: Vec<AssetOptionalPrice>,
}
```

**Listing 2.7:** common/src/lib.rs

It is recommended to add a reasonable threshold to check the freshness of `PriceData` received from the price oracle.

```rust
impl From<PriceData> for Prices {
  fn from(data: PriceData) -> Self {
    Self {
      prices: data
      .prices
      .into_iter()
      .filter_map(|AssetOptionalPrice { asset_id, price }| {
        price.map(|price| (asset_id, price))
      })
      .collect(),
    }
  }
}
```

**Listing 2.8:** contract/src/prices.rs

**Impact** The received `PriceDate` may be outdated.

**Suggestion I** Add a reasonable threshold to check the freshness of `PriceData`.

### 2.2.5 The configuration should be checked in init function

**Status** Fixed.

**Description** This issue is introduced in and before the version 1.0. The validation of the config needs to be checked in the initialization function `new`.

```rust
#[init]
pub fn new(config: Config) -> Self {
  Self {
    accounts: UnorderedMap::new(StorageKey::Accounts),
    storage: LookupMap::new(StorageKey::Storage),
    assets: LookupMap::new(StorageKey::Assets),
    asset_farms: LookupMap::new(StorageKey::AssetFarms),
    asset_ids: UnorderedSet::new(StorageKey::AssetIds),
    config: LazyOption::new(StorageKey::Config, Some(&config)),
  }
}
```

**Listing 2.9:** contract/src/lib.rs

**Impact** The configuration may be invalid while initializing the contract.

**Suggestion I** Check the configuration with `config.assert_valid()` .

## 2.3 Additional Recommendation

### 2.3.1 Dead code is found

**Status** Fixed.

**Description** This issue is introduced in or before the initial commit. Function `init` is not used.

```
fn init(&mut self) {
  self.initial_storage_usage = env::storage_usage();
}
```

**Listing 2.10:** contract/src/storage.rs

**Suggestion I** Remove the function `init` in `storage.rs`.

### 2.3.2 Deflation Token is not supported

**Status** Confirmed.

**Description** This issue is introduced in or before the initial commit. Currently, the contract only supports NEP141 token. Other tokens like deflation tokens are not supported. In this case, the contract owner should be careful when adding new asset tokens.

### 2.3.3 Redundant invoked functions

**Status** Fixed.

**Description** This issue is introduced in or before the initial commit. The function `env::predecessor_account_id()` is invoked and assigned to the variable `account_id` in line 219. In this case, we can use the variable `account_id` in line 221 instead of invoking function `env::predecessor_account_id()` again.

```
218  pub fn account_farm_claim_all(&mut self) {
219    let account_id = env::predecessor_account_id();
220    let (mut account, storage) =
221    self.internal_unwrap_account_with_storage(&env::predecessor_account_id());
222    account.add_all_affected_farms();
223    self.internal_account_apply_affected_farms(&mut account, false);
224    self.internal_set_account(&account_id, account, storage);
225  }
```

**Listing 2.11:** contract/src/storage.rs

**Suggestion I** Replace the `env::predecessor_account_id` by variable `account_id`.

### 2.3.4 Reliability of oracle

**Status** Confirmed.

**Description** This issue is introduced in or before the initial commit. The price of different tokens is fed by the external oracle contract. Actions including `DecreaseCollateral`, `Borrow`, `Liquidate` all depend on the token prices. In this case, a stable and reliable oracle is a requirement.

### 2.3.5 Upgrade module of the contract

**Status** Fixed.

**Description** This issue is introduced in or before the initial commit. Currently, no upgrade module is implemented. When the code changes but the serialized state stays the same, the contract may not work. It is recommended to add the upgrade module for further upgrade.

### 2.3.6 The risk of centralized design

**Status** Confirmed

**Description** This issue is introduced in or before the initial commit. The project has a highly centralized design. The contract owner has very high privilege that can update the asset config, contract config, add the asset farm reward, etc. We highly suggest that the project owner should enforce security mechanisms to protect the private keys of the contract owner to manage the contracts.

### 2.3.7 Allowed actions

**Status** Acknowledged

**Description** This issue is introduced in or before the initial commit. In this contract, there are three functions (i.e., `execute()`, `oracle_on_call()`, `ft_on_transfer()`) for executing the actions. Among the three functions, actions that rely on prices should be executed by invoking `oracle_on_call()`. We suggest to check the allowed actions in each function to reduce the attack surface. Apart from this, it would also be more safe to ensure that the executed actions are not repeated.

### 2.3.8 Conflicts in Documentation

**Status** Fixed

**Description** This issue is introduced in or before the version 1.0. In the documentation (i.e., README.md), it says "Bob gets farming multiplier of 2X, so the number of boosted shares becomes 100 (50 * `3`)". The correct description is "Bob gets farming multiplier of 2X, so the number of boosted shares becomes 100 (50 * `2`)". Apart from this, it says "Let's say there is a farm for supplying USDC that gives `1000` wNEAR per day". The correct description is "Let's say there is a farm for supplying USDC that gives `200` wNEAR per day".