HALBORN

Parallel.fi - Money Market

Substrate Pallet Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 29th, 2022 - May 28th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/29/2022	Hossam Mohamed
0.2	Document Edits	05/28/2022	Hossam Mohamed
0.3	Document Edits	06/03/2022	Hossam Mohamed
0.4	Draft Review	06/03/2022	Timur Guvenkaya
0.5	Draft Final Review	06/03/2022	Gabi Urrutia
1.0	Remediation Plan	06/13/2022	Hossam Mohamed
1.1	Remediation Plan Edits	06/14/2022	Timur Guvenkaya
1.2	Remediation Plan Review	06/17/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Timur Guvenkaya	Halborn	Timur.Guvenkaya@halborn.com
Hossam Mohamed	Halborn	hossam.mohamed@halborn.com
Thiago Mathias	Halborn	Thiago.Mathias@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Parallel.fi engaged Halborn to conduct a security assessment on their Money Market (loans) Substrate pallet on April 29th, 2022 and ending May 28th, 2022. Parallel.fi is the decentralized platform that empowers everyone access to financial services built on Substrate.

1.2 AUDIT SUMMARY

The team at Halborn was provided 3 weeks for the engagement and assigned two full-time security engineers to audit the security of the assets in scope. The engineers are blockchain and smart contract security experts with advanced penetration testing, smart-contract hacking, and in-depth knowledge of multiple blockchain protocols.

The purpose of this audit is to achieve the following:

- Identify potential security issues within the substrate Money Market (loans) pallet.

In summary, Halborn identified few security risks that were addressed by Parallel.fi team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the MM Substrate pallet. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Substrate Pallets manual code review and walkthrough
- Mapping out possible attack vectors
- On chain testing of core functions.
- Fuzzing of core functions through cargo-fuzz
- - Fuzzing of core functions through honggfuzz
- Finding security vulnerabilities through cargo-audit and cargo-deny
- Finding usage of unsafe Rust within the project through cargo-geiger
- Testnet deployment (polkadot.js)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.

3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

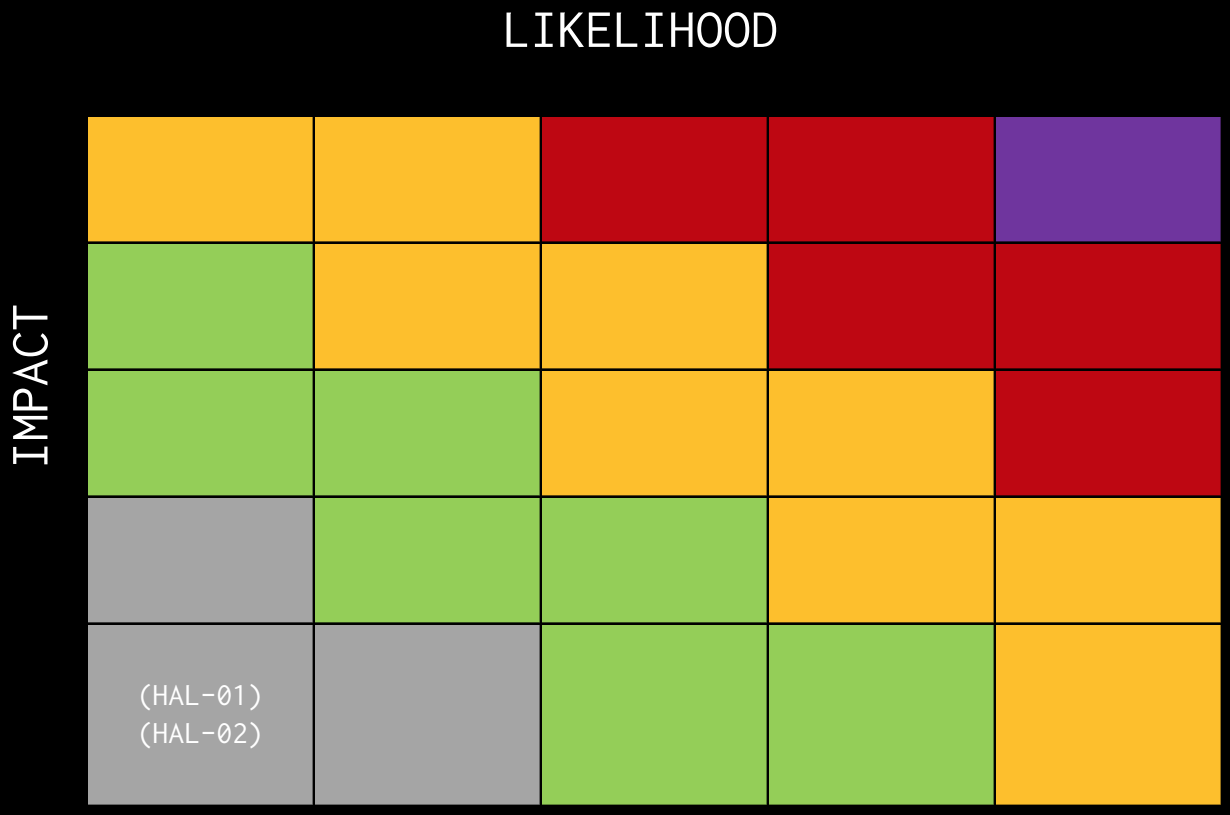
1.4 SCOPE

The review was scoped to the MM Substrate pallet in the mm-farm-audit branch in parallel-finance repository.

- Pallets
  - Money Market (loans)

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	2



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
THRESHOLDS MIN VALUES NOT ENFORCED	Informational	NOT APPLICABLE
MISSING ZERO CHECK	Informational	NOT APPLICABLE

FINDINGS & TECH DETAILS

3.1 (HAL-01) THRESHOLDS MIN VALUES NOT ENFORCED - INFORMATIONAL

Description:

The update_market function from the **loans** pallet modifies the close_factor variable without proper validation, allowing for values lesser than the minimum expected.

The aforementioned value is used to define the maximum liquidation limit at a time, if the close_factor variable takes the value zero, no user will be able to liquidate their borrowings.

Code Location:

Listing 1: pallets/loans/src/lib.rs (Line 588)

```
559 pub fn update_market(
560     origin: OriginFor<T>,
561     asset_id: AssetIdOf<T>,
562     collateral_factor: Ratio,
563     reserve_factor: Ratio,
564     close_factor: Ratio,
565     liquidate_incentive: Rate,
566     #[pallet::compact] supply_cap: BalanceOf<T>,
567     #[pallet::compact] borrow_cap: BalanceOf<T>,
568 ) -> DispatchResultWithPostInfo {
569     T::UpdateOrigin::ensure_origin(origin)?;
570 
571     ensure!(
572         collateral_factor > Ratio::zero() && collateral_factor <
 ↳ Ratio::one(),
573         Error::<T>::InvalidFactor
574     );
575     ensure!(
576         reserve_factor > Ratio::zero() && reserve_factor < Ratio::
 ↳ one(),
577         Error::<T>::InvalidFactor
578     );
579     ensure!(supply_cap > Zero::zero(), Error::<T>::
```

```
 ↳ InvalidSupplyCap);
580
581     let market = Self::mutate_market(asset_id, |stored_market| {
582         *stored_market = Market {
583             state: stored_market.state,
584             ptoken_id: stored_market.ptoken_id,
585             rate_model: stored_market.rate_model,
586             collateral_factor,
587             reserve_factor,
588             close_factor,
589             liquidate_incentive,
590             supply_cap,
591             borrow_cap,
592         };
593         stored_market.clone()
594     })?;
595     Self::deposit_event(Event::<T>::UpdatedMarket(market));
596
597     Ok(().into())
598 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

A validation routine should be added inside the update_market function to enforce that the values of close_factor are within the expected range.

Remediation Plan:

NOT APPLICABLE: The Parallel team marked the issue as not applicable, as the close factor is set at 50% and non issue.

3.2 (HAL-02) MISSING ZERO CHECK - INFORMATIONAL

Description:

There are no validations to check if borrow_amount is equal to zero, allowing to borrow zero.

Code Location:

Listing 2: parallel/pallets/loans/src/lib.rs (Line 848)

```
845 pub fn borrow(
846     origin: OriginFor<T>,
847     asset_id: AssetIdOf<T>,
848     #[pallet::compact] borrow_amount: BalanceOf<T>,
849 ) -> DispatchResultWithPostInfo {
850     let who = ensure_signed(origin)?;
851     Self::ensure_active_market(asset_id)?;
852
853     Self::borrow_allowed(asset_id, &who, borrow_amount)?;
854     Self::accrue_interest(asset_id)?;
855
856     // update borrow index after accureInterest.
857     Self::update_reward_borrow_index(asset_id)?;
858     Self::distribute_borrower_reward(asset_id, &who)?;
859
860     let account_borrows = Self::current_borrow_balance(&who,
 ↳ asset_id)?;
861     let account_borrows_new = account_borrows
862         .checked_add(borrow_amount)
863         .ok_or(ArithmeticError::Overflow)?;
864     let total_borrows = Self::total_borrows(asset_id);
865     let total_borrows_new = total_borrows
866         .checked_add(borrow_amount)
867         .ok_or(ArithmeticError::Overflow)?;
868     AccountBorrows::<T>::insert(
869         asset_id,
870         &who,
871         BorrowSnapshot {
872             principal: account_borrows_new,
```

```
873             borrow_index: Self::borrow_index(asset_id),
874         },
875     );
876     TotalBorrows::<T>::insert(asset_id, total_borrows_new);
877     T::Assets::transfer(asset_id, &Self::account_id(), &who,
 ↳ borrow_amount, false)?;
878
879     Self::deposit_event(Event::<T>::Borrowed(who, asset_id,
 ↳ borrow_amount));
880     Ok(().into())
881 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Implement validation to check if the value of borrow_amount is equal to zero.

Remediation Plan:

NOT APPLICABLE: The Parallel team marked the issue as not applicable, as they don't allow 0 borrow from front end.

THANK YOU FOR CHOOSING

// HALBORN