HALBORN

Composable.Finance - Pablo

Substrate Pallet Security Audit

Prepared by: **Halborn**

Date of Engagement: **July 30th, 2022 - August 17th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/08/2022	Alpcan Onaran
0.2	Document Edits	08/18/2022	Alpcan Onaran
0.3	Document Edits	08/19/2022	Alpcan Onaran
0.4	Draft Review	08/20/2022	Timur Guvenkaya
0.5	Draft Review	08/22/2022	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
Alpcan Onaran	Halborn	Alpcan.Onaran@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Composable engaged Halborn to conduct a security assessment on their main Substrate pallets on July 30th, 2022 and ending August 17th, 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 2.5 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 Pablo pallet.

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 lead 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.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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 frame/pablo directory using 495 faa2a132654cafb10ed55bf4eee0446261ef0 commit-id in ComposableFi/composable repository.

- Pallets
  - Pablo
  -
    - Helper pallet functions

EXECUTIVE OVERVIEW

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	0	3	3

LIKELIHOOD

IMPACT

🟨	🟨	🟥	🟥	🟪
🟩	🟨	🟨	🟥 (HAL-01)	🟥
🟩	🟩	🟨	🟨	🟥
⬜ (HAL-05)	🟩 (HAL-02) (HAL-03) (HAL-04)	🟩	🟨	🟨
⬜ (HAL-06) (HAL-07)	⬜	🟩	🟩	🟨

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 ACCOUNTS CAN CREATE SAME PAIR POOLS WITHOUT LIMITS	High	-
HAL-02 MISSING PAUSABLE FUNCTIONALITY	Low	-
HAL-03 ZERO AMOUNT BUY-SELL-SWAP	Low	-
HAL-04 ZERO AMOUNT REMOVE LIQUIDITY	Low	-
HAL-05 CREATING STABLE SWAP WITH NON-STABLE TOKENS	Informational	-
HAL-06 MISLEADING ERROR	Informational	-
HAL-07 CREATING POOLS ON BEHALF OF OTHER ACCOUNTS	Informational	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) USERS CAN CREATE SAME PAIR POOLS WITHOUT LIMITS - HIGH

Description:

It was observed that, an account can create configured pools without any limitation, it was also observed that there are not any functionality to remove pools for StableSwap and ConstantProduct Pools, attackers can use this functionality to create pools with same configuration in a loop and fill up the PoolCount u16 variable, which will prevent further pool creation for other users.

It was calculated that attacker can perform this attack using 327670000 weight.

Code Location:

Listing 1: frame/pablo/src/lib.rs

```
    #[transactional]
    pub fn do_create_pool(
        init_config: PoolInitConfigurationOf<T>,
    ) -> Result<T::PoolId, DispatchError> {
        let (owner, pool_id, pair) = match init_config {
            PoolInitConfiguration::StableSwap {
                owner,
                pair,
                amplification_coefficient,
                fee,
            } => {
                let pool_id = StableSwap::<T>::do_create_pool(
                    &owner,
                    pair,
                    amplification_coefficient,
                    FeeConfig::default_from(fee),
                )?;
                Self::create_staking_reward_pool(&pool_id, pair)?;
                (owner, pool_id, pair)
            },
        PoolInitConfiguration::ConstantProduct { owner, pair, fee, base_weight } => {
          let pool_id = Uniswap::<T>::do_create_pool(
            &owner,
            pair,
            FeeConfig::default_from(fee),
            base_weight,
          )?;
          Self::create_staking_reward_pool(&pool_id, pair)?;
          (owner, pool_id, pair)
        },
        PoolInitConfiguration::LiquidityBootstrapping(pool_config) => {
          let validated_pool_config =
            Validated::new(pool_config.clone()).map_err(DispatchError::Other)?;
          (
            pool_config.owner,
            LiquidityBootstrapping::<T>::do_create_pool(validated_pool_config)?,
            pool_config.pair,
          )
        },
      };
      Self::deposit_event(Event::<T>::PoolCreated { owner, pool_id, assets: pair });
      Ok(pool_id)
```

Proof Of Concept:

Listing 2: Stopping-Pool-Creation (Line 10)

```
1 #[test]
2 fn same_pair_pool_creation_dos() {
3   new_test_ext().execute_with(|| {
4     let pool_init_config = PoolInitConfiguration::StableSwap {
5       owner: ALICE,
6       pair: CurrencyPair::new(USDC, USDT),
7       amplification_coefficient: 100_u16,
8       fee: Permill::zero(),
9     };
```

```
        for i in 0..u16::MAX {
        let pool_id = Pablo::do_create_pool(pool_init_config.clone()).expect("pool creation failed");
        }
        //Pool creation stops after this for other users because PoolCount overflows.
    });
}
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

It is recommended to ensure that an account cannot create a same type pool with same pairs. It is also recommended to add a function to remove pools while returning the liquidity users.

3.2 (HAL-02) MISSING PAUSABLE FUNCTIONALITY - LOW

Description:

It was observed that, pools does not have a pausable functionality to protect the users from possible abnormal situations.

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to add an admin controlled pausable functionality to pools to protect users from unexpected situations.

3.3 (HAL-03) ZERO AMOUNT BUY-SELL-SWAP - LOW

Description:

It was observed that, buy, sell and swap functions does not check if the amount equals to zero. Zero amount of wrappings can be abused if someone constantly calls these functions with zero amount and fill the block space, which may delay or halt other user transactions.

Code Location:

Listing 3: frame/pablo/src/lib.rs (Line 453)

```
    /// Execute a buy order on pool.
    ///
    /// Emits `Swapped` event when successful.
    #[pallet::weight(T::WeightInfo::buy())]
    pub fn buy(
        origin: OriginFor<T>,
        pool_id: T::PoolId,
        asset_id: T::AssetId,
        amount: T::Balance,
        min_receive: T::Balance,
        keep_alive: bool,
    ) -> DispatchResult {
        let who = ensure_signed(origin)?;
        let _ = <Self as Amm>::buy(&who, pool_id, asset_id, amount, min_receive, keep_alive)?;
        Ok(())
    }

    /// Execute a sell order on pool.
    ///
    /// Emits `Swapped` event when successful.
    #[pallet::weight(T::WeightInfo::sell())]
    pub fn sell(
        origin: OriginFor<T>,
        pool_id: T::PoolId,
        asset_id: T::AssetId,
```

```rust
            amount: T::Balance,
            min_receive: T::Balance,
            keep_alive: bool,
        ) -> DispatchResult {
            let who = ensure_signed(origin)?;
            let _ = <Self as Amm>::sell(&who, pool_id, asset_id, amount, min_receive, keep_alive)?;
            Ok(())
        }

        /// Execute a specific swap operation.
        ///
        /// The `quote_amount` is always the quote asset amount (A/B => B), (B/A => A).
        ///
        /// Emits `Swapped` event when successful.
        #[pallet::weight(T::WeightInfo::swap())]
        pub fn swap(
            origin: OriginFor<T>,
            pool_id: T::PoolId,
            pair: CurrencyPair<T::AssetId>,
            quote_amount: T::Balance,
            min_receive: T::Balance,
            keep_alive: bool,
        ) -> DispatchResult {
            let who = ensure_signed(origin)?;
            let _ = <Self as Amm>::exchange(
                &who,
                pool_id,
                pair,
                quote_amount,
                min_receive,
                keep_alive,
            )?;
            Ok(())
        }
```

Proof Of Concept:

Listing 4: Zero-Amount-Buy-Sell-Swap (Lines 29,30,31)

```
 1 #[test]
 2 fn zero_buy_sell_swap_stableswap() {
 3     new_test_ext().execute_with(|| {
 4         let pool_init_config = PoolInitConfiguration::StableSwap {
 5             owner: ALICE,
 6             pair: CurrencyPair::new(USDC, USDT),
 7             amplification_coefficient: 100_u16,
 8             fee: Permill::from_percent(10),
 9         };
10         let pool_id = Pablo::do_create_pool(pool_init_config).
 ↳ expect(”pool creation failed”);
11         let pool = Pablo::pools(pool_id).expect(”pool not found”);
12         let pool = match pool {
13             StableSwap(pool) => pool,
14             _ => panic!(”expected stable_swap pool”),
15         };
16
17         // Mint the tokens
18         assert_ok!(Tokens::mint_into(USDC, &ALICE, 1));
19         assert_ok!(Tokens::mint_into(USDT, &ALICE, 1));
20         // Add the liquidity
21         assert_ok!(Pablo::add_liquidity(
22             Origin::signed(ALICE),
23             pool_id,
24             1,
25             1,
26             0,
27             false
28         ));
29         Pablo::sell(Origin::signed(BOB), pool_id, USDC, 0, 0_u128,
 ↳  false).expect(”sell failed”);
30         Pablo::buy(Origin::signed(BOB), pool_id, USDC, 0, 0_u128,
 ↳ false).expect(”sell failed”);
31         assert_ok!(Pablo::swap(Origin::signed(BOB), pool_id,
 ↳ CurrencyPair::new(USDC, USDT), 0, 0, false));
32
33     });
34 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to ensure that input parameter amount is higher than zero.

3.4 (HAL-04) ZERO AMOUNT REMOVE LIQUIDITY - LOW

Description:

It was observed that, remove_liquidity function does not check if the lp_amount equals to zero. Zero amount of wrappings can be abused if someone constantly calls remove_liquidity with zero amount and fill the block space, which may delay or halt other user transactions.

Code Location:

Listing 5: frame/pablo/src/lib.rs

```
        #[transactional]
        fn remove_liquidity(
            who: &Self::AccountId,
            pool_id: Self::PoolId,
            lp_amount: Self::Balance,
            min_base_amount: Self::Balance,
            min_quote_amount: Self::Balance,
        ) -> Result<(), DispatchError> {
            let currency_pair = Self::currency_pair(pool_id)?;
            let redeemable_assets = Self::redeemable_assets_for_lp_tokens(
                pool_id,
                lp_amount,
                BTreeMap::from([
                    (currency_pair.base, min_base_amount),
                    (currency_pair.quote, min_quote_amount),
                ]),
            )?;
            let pool = Self::get_pool(pool_id)?;
            let pool_account = Self::account_id(&pool_id);
            match pool {
...
        }
```

Proof Of Concept:

Listing 6: Zero-Amount-RemoveLiquidity (Line 33)

```
 1 #[test]
 2 fn zero_amount_remove_liquidity() {
 3     new_test_ext().execute_with(|| {
 4         let pool_init_config = PoolInitConfiguration::
 ↳ ConstantProduct {
 5             owner: ALICE,
 6             pair: CurrencyPair::new(BTC, USDT),
 7             fee: Permill::zero(),
 8             base_weight: Permill::from_percent(50),
 9         };
10         let pool_id = Pablo::do_create_pool(pool_init_config).
 ↳ expect("pool creation failed");
11
12         let pool = get_pool(pool_id);
13
14         let current_product = |a| {
15             let balance_btc = Tokens::balance(BTC, &a);
16             let balance_usdt = Tokens::balance(USDT, &a);
17             balance_btc * balance_usdt
18         };
19
20         // Mint the tokens
21         assert_ok!(Tokens::mint_into(BTC, &ALICE, 1));
22         assert_ok!(Tokens::mint_into(USDT, &ALICE, 1));
23
24         // Add the liquidity
25         assert_ok!(<Pablo as Amm>::add_liquidity(
26             &ALICE,
27             pool_id,
28             1,
29             1,
30             0,
31             false
32         ));
33         assert_ok!(<Pablo as Amm>::remove_liquidity(&ALICE,
 ↳ pool_id, 0, 0, 0));
34     });
35 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to ensure input parameter lp_amount is higher than zero in remove_liquidity function.

3.5 (HAL-05) CREATING STABLE SWAP WITH NON-STABLE TOKENS - INFORMATIONAL

Description:

It was observed that users can start stable swap pools using any pair, including non-stable tokens (Example: wBTC-USDT).

Code Location:

Listing 7: frame/pablo/src/stable_swap.rs (Line 24)

```
22      pub fn do_create_pool(
23          who: &T::AccountId,
24          pair: CurrencyPair<T::AssetId>,
25          amp_coeff: u16,
26          fee: FeeConfig,
27      ) -> Result<T::PoolId, DispatchError> {
28          ensure!(amp_coeff > 0, Error::<T>::
 ↳ AmpFactorMustBeGreaterThanZero);
29          ensure!(pair.base != pair.quote, Error::<T>::InvalidPair);
30          ensure!(fee.fee_rate < Permill::one(), Error::<T>::
 ↳ InvalidFees);
31
32          let lp_token = T::CurrencyFactory::create(RangeId::
 ↳ LP_TOKENS, T::Balance::default())?;
33          // Add new pool
34          let pool_id =
35              PoolCount::<T>::try_mutate(|pool_count| -> Result<T::
 ↳ PoolId, DispatchError> {
36                  let pool_id = *pool_count;
37
38                  Pools::<T>::insert(
39                      pool_id,
40                      PoolConfiguration::StableSwap(
 ↳ StableSwapPoolInfo {
41                          owner: who.clone(),
42                          pair,
43                          lp_token,
                        amplification_coefficient: amp_coeff,
                        fee_config: fee,
                    }),
                );
                *pool_count = pool_id.safe_add(&T::PoolId::one())?;
                Ok(pool_id)
            })?;

        Ok(pool_id)
    }
```

Proof Of Concept:

Listing 8: StableSwap-NonStableTokens (Line 6)

```
#[test]
fn stableswap_with_nonstabletokens() {
    new_test_ext().execute_with(|| {
        let pool_init_config = PoolInitConfiguration::StableSwap {
            owner: ALICE,
            pair: CurrencyPair::new(BTC, USDT),
            amplification_coefficient: 100_u16,
            fee: Permill::zero(),
        };
        let pool_id = Pablo::do_create_pool(pool_init_config).expect("pool creation failed");
        let pool = Pablo::pools(pool_id).expect("pool not found");
        let pool = match pool {
            StableSwap(pool) => pool,
            _ => panic!("expected stable_swap pool"),
        };

        let btc_price = 23200;
        let nb_of_btc = 100;
        let usdt_price = 1;
        let nb_of_usdt = 1000000;

        let initial_btc = nb_of_btc * btc_price;
        let initial_usdt = nb_of_usdt * usdt_price;

        // Mint the tokens
        assert_ok!(Tokens::mint_into(BTC, &ALICE, initial_btc));
        assert_ok!(Tokens::mint_into(USDT, &ALICE, initial_usdt));

        // Add the liquidity
        assert_ok!(Pablo::add_liquidity(
            Origin::signed(ALICE),
            pool_id,
            initial_btc,
            initial_usdt,
            0,
            false
        ));

        // 1 unit of usdc == 1 unit of usdt
        let ratio = <Pablo as Amm>::get_exchange_value(pool_id, BTC, 1)
            .expect("get_exchange_value failed");
        println!("{:?}", ratio);
        assert_ok!(Tokens::mint_into(BTC, &ALICE, initial_btc));
        let ratio = <Pablo as Amm>::get_exchange_value(pool_id, BTC, 1)
            .expect("get_exchange_value failed");
        println!("{:?}", ratio);
    });
}
```

Risk Level:

Likelihood - 1

Impact - 2

Recommendation:

Our team does not find any direct impact thought this functionality but since StableSwap pools are used for stable token swaps, it is recommended to restrict the non-stable token assets in StablePools.

3.6 (HAL-06) MISLEADING ERROR - INFORMATIONAL

Description:

Inside composable-maths/src/dex/constant_product.rs, it was observed that compute_out_given_in and compute_out_given_out functions are using a misleading error message, the functions throwing a ArithmeticError ::Overflow message when the sum of wi and wo varilables exceeds the expected amount. ArithmeticError::Overflow should be used when an integer overflow/underflow vulnerability gets detected.

Code Location:

Listing 9: composable-maths/src/dex/constant_product.rs (Line 73)

```
59 pub fn compute_out_given_in<T: PerThing>(
60     wi: T,
61     wo: T,
62     bi: u128,
63     bo: u128,
64     ai: u128,
65 ) -> Result<u128, ArithmeticError>
66 where
67     T::Inner: Into<u32>,
68 {
69     let wi: u32 = wi.deconstruct().into();
70     let wo: u32 = wo.deconstruct().into();
71     let weight_sum = wi.safe_add(&wo)?;
72     let expected_weight_sum: u32 = T::one().deconstruct().into();
73     ensure!(weight_sum == expected_weight_sum, ArithmeticError::
 ↳ Overflow);
74     ...
```

Listing 10: composable-maths/src/dex/constant_product.rs (Line 110)

```
96 pub fn compute_in_given_out<T: PerThing>(
97     wi: T,
98     wo: T,
    bi: u128,
    bo: u128,
    ao: u128,
) -> Result<u128, ArithmeticError>
where
    T::Inner: Into<u32>,
{
    let wi: u32 = wi.deconstruct().into();
    let wo: u32 = wo.deconstruct().into();
    let weight_sum = wi.safe_add(&wo)?;
    let expected_weight_sum: u32 = T::one().deconstruct().into();
    ensure!(weight_sum == expected_weight_sum, ArithmeticError::Overflow);
    ...
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to change the error message to prevent possible misleading errors.

3.7 (HAL-07) CREATING POOLS ON BEHALF OF OTHER ACCOUNTS - INFORMATIONAL

Description:

It was observed that users can start pool on behalf of other accounts.

Code Location:

Listing 11: frame/pablo/src/lib.rs (Line 748)

```
744         #[transactional]
745         pub fn do_create_pool(
746             init_config: PoolInitConfigurationOf<T>,
747         ) -> Result<T::PoolId, DispatchError> {
748             let (owner, pool_id, pair) = match init_config {
749                 PoolInitConfiguration::StableSwap {
750                     owner,
751                     pair,
752                     amplification_coefficient,
753                     fee,
754                 } => {
755                     let pool_id = StableSwap::<T>::do_create_pool(
756                         &owner,
757                         pair,
758                         amplification_coefficient,
759                         FeeConfig::default_from(fee),
760                     )?;
761                     Self::create_staking_reward_pool(&pool_id, pair)?;
762                     (owner, pool_id, pair)
763                 },
764                 PoolInitConfiguration::ConstantProduct { owner, pair, fee, base_weight } => {
765                     let pool_id = Uniswap::<T>::do_create_pool(
766                         &owner,
767                         pair,
768                         FeeConfig::default_from(fee),
769                         base_weight,
                    )?;
                    Self::create_staking_reward_pool(&pool_id, pair)?;
                    (owner, pool_id, pair)
                },
                PoolInitConfiguration::LiquidityBootstrapping(pool_config) => {
                    let validated_pool_config =
                        Validated::new(pool_config.clone()).map_err(DispatchError::Other)?;
                    (
                        pool_config.owner,
                        LiquidityBootstrapping::<T>::do_create_pool(validated_pool_config)?,
                        pool_config.pair,
                    )
                },
            };
            Self::deposit_event(Event::<T>::PoolCreated { owner, pool_id, assets: pair });
            Ok(pool_id)
        }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Even this functionality will be used to transfer pools to ComposableFi side for them to manage, it is recommended to limit the parameter owner to specific accounts only to prevent possible griefing attacks.

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: chrono
Version: 0.4.19
Title: Potential segfault in localtime_r invocations
Date: 2020-11-10
ID: RUSTSEC-2020-0159
URL: https://rustsec.org/advisories/RUSTSEC-2020-0159
Solution: Upgrade to >=0.4.20

Crate: hyper
Version: 0.10.16
Title: Lenient hyper header parsing of Content-Length could allow request smuggling & Integer overflow in hyper's parsing of the Transfer-Encoding header leads to data loss
Date: 2021-07-07
ID: RUSTSEC-2021-0078 & RUSTSEC-2021-0079
URL: https://rustsec.org/advisories/RUSTSEC-2021-0078 & https://rustsec.org/advisori
2021-0078
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

Crate:  rocksdb
Version:  0.18.0
Title:  Out-of-bounds read when opening multiple column families with TTL
Date:  2022-05-11
ID: RUSTSEC-2022-0046
URL: https://rustsec.org/advisories/RUSTSEC-2022-0046
Solution:  Upgrade to >=0.19.0

Crate:  websocket
Version:  0.24.0
Title:  Unbounded memory allocation based on untrusted length
Date:  2022-08-01
ID: RUSTSEC-2022-0035
URL: https://rustsec.org/advisories/RUSTSEC-2022-0035
Solution:  Upgrade to >=0.26.5

Crate:  aes-soft
Version:  0.6.4
Warning:  unmaintained
Title:  aes-soft has been merged into the aes crate
Date:  2021-04-29
ID: RUSTSEC-2021-0060
URL: https://rustsec.org/advisories/RUSTSEC-2021-0060

Crate:  aesni
Version:  0.10.0
Warning:  unmaintained
Title:  aesni has been merged into the aes crate
```

```
Date:  2021-04-29
ID: RUSTSEC-2021-0059
URL: https://rustsec.org/advisories/RUSTSEC-2021-0059

Crate:  ansi_term
Version:  0.12.1
Warning:  unmaintained
Title:  ansi_term is Unmaintained
Date:  2021-08-18
ID: RUSTSEC-2021-0139
URL: https://rustsec.org/advisories/RUSTSEC-2021-0139

Crate:  cpuid-bool
Version:  0.2.0
Warning:  unmaintained
Title:  cpuid-bool has been renamed to cpufeatures
Date:  2021-05-06
ID: RUSTSEC-2021-0064
URL: https://rustsec.org/advisories/RUSTSEC-2021-0064

Crate:  net2
Version:  0.2.37
Warning:  unmaintained
Title:  net2 crate has been deprecated; use socket2 instead
Date:  2020-05-01
ID: RUSTSEC-2020-0016
URL: https://rustsec.org/advisories/RUSTSEC-2020-0016

Crate:  stdweb
Version:  0.4.20
Warning:  unmaintained
Title:  stdweb is unmaintained
Date:  2020-05-04
ID: RUSTSEC-2020-0056
URL: https://rustsec.org/advisories/RUSTSEC-2020-0056
```

THANK YOU FOR CHOOSING

// HALBORN