// HALBORN

DRAFT

Composable.Finance - Vesting

Substrate Pallet Security Audit

Prepared by: **Halborn**

Date of Engagement: **August 23rd, 2022 - September 5th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	09/03/2022	Michal Bajor
0.2	Document Edits	09/04/2022	Michal Bajor
0.3	Draft Review	09/05/2022	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.Finance engaged Halborn to conduct a security audit on their smart contracts beginning on August 23rd, 2022 and ending on September 5th, 2022 The security assessment was scoped to the smart contracts provided to the Halborn team.

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:

- Ensure that Substrate pallet's functions operate as intended
- Identify potential security issues with the Substrate pallet

In summary, Halborn identified some security risks that should be addressed by the Composable.Finance 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.
- 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

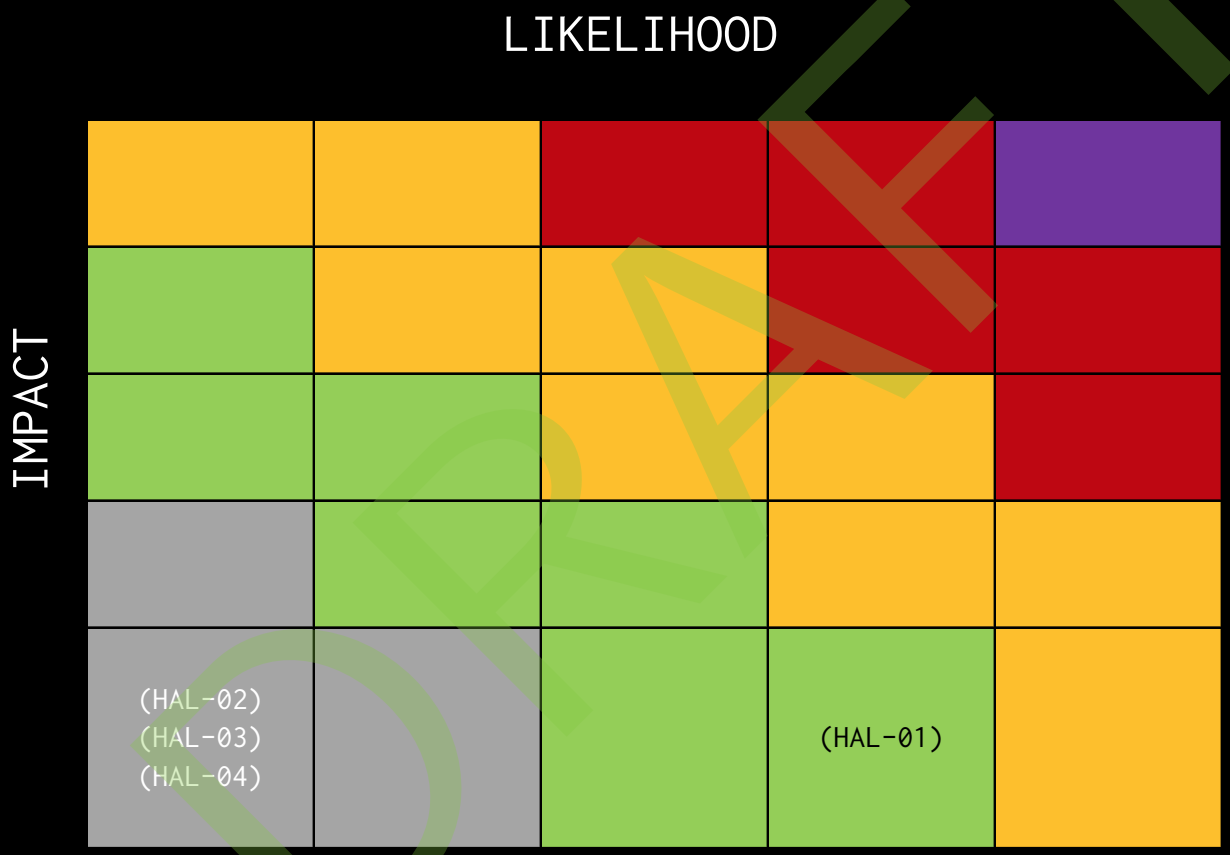
The review was scoped to the vesting pallet in frame directory in the ComposableFi/composable repository.

Commit IDs used for the engagement:

- 5dd44ed800225e06eda72b297311e25221e036f4

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	3



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
POSSIBILITY OF CREATING A VESTING SCHEDULE IN THE PAST	Low	-
USAGE OF DEPRECATED MACRO	Informational	-
USAGE OF ROOT ORIGIN	Informational	-
USAGE OF SUDO PALLET	Informational	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) POSSIBILITY OF CREATING A VESTING SCHEDULE IN THE PAST - LOW

Description:

The vesting pallet defines a vested_transfer extrinsic, which is responsible for creating a vesting schedule. This function does not verify the timeline of the schedule. As a consequence, it is possible to create a vesting schedule in the past, i.e., the one that is already finished when created.

Code Location:

A vested_transfer extrinsic:

Listing 1: frame/vesting/src/lib.rs (Lines 338,343)

```
332 #[pallet::weight(<T as Config>::WeightInfo::vested_transfer())]
333 pub fn vested_transfer(
334     origin: OriginFor<T>,
335     from: <T::Lookup as StaticLookup>::Source,
336     beneficiary: <T::Lookup as StaticLookup>::Source,
337     asset: AssetIdOf<T>,
338     schedule_info: VestingScheduleInfoOf<T>,
339 ) -> DispatchResult {
340     T::VestedTransferOrigin::ensure_origin(origin)?;
341     let from = T::Lookup::lookup(from)?;
342     let to = T::Lookup::lookup(beneficiary)?;
343     <Self as VestedTransfer>::vested_transfer(asset, &from, &to,
 ↳ schedule_info)?;
344
345     Ok(())
346 }
```

An internal vested_transfer function called by vested_transfer extrinsic:

Listing 2: frame/vesting/src/lib.rs (Lines 422,434-438)

```
412 #[transactional]
413 fn vested_transfer(
414     asset: Self::AssetId,
415     from: &Self::AccountId,
416     to: &Self::AccountId,
417     schedule_info: VestingScheduleInfo<Self::BlockNumber, Self::
 ↳ Moment, Self::Balance>,
418 ) -> frame_support::dispatch::DispatchResult {
419     ensure!(from != to, Error::<T>::TryingToSelfVest);
420
421     let vesting_schedule_id = Self::VestingScheduleNonce::
 ↳ increment()?;
422     let schedule = VestingSchedule::from_input(vesting_schedule_id
 ↳ , schedule_info);
423
424     let schedule_amount = ensure_valid_vesting_schedule::<T>(&
 ↳ schedule)?;
425
426     let locked = Self::locked_balance(to, asset,
 ↳ VestingScheduleIdSet::All)
427         .unwrap_or_else(|_| Zero::zero());
428
429     let total_amount = locked.safe_add(&schedule_amount)?;
430
431     T::Currency::transfer(asset, from, to, schedule_amount)?;
432     T::Currency::set_lock(VESTING_LOCK_ID, asset, to, total_amount
 ↳ )?;
433
434     <VestingSchedules<T>>::mutate(to, asset, |schedules| {
435         schedules
436             .try_insert(vesting_schedule_id, schedule.clone())
437             .map_err(|_| Error::<T>::MaxVestingSchedulesExceeded)
438     })?;
439
440     Self::deposit_event(Event::VestingScheduleAdded {
441         from: from.clone(),
442         to: to.clone(),
443         asset,
444         schedule,
445         vesting_schedule_id,
446     });
447
448     Ok(())
```

```
449 }
```

Risk Level:

Likelihood - 4
Impact - 1

Recommendation:

It is recommended to implement a validation mechanism responsible for making sure that the timeline of the vesting schedule is not set in the past.

3.2 (HAL-02) USAGE OF DEPRECATED MACRO - INFORMATIONAL

Description:

The vesting pallet is built with the Polkadot version 0.9.27. This version implements the #[transactional] behaviour by default for every extrinsic, making that macro obsolete.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to remove the usage of deprecated #[transactional] macro.

Reference:

- #[transactional] macro behavior is implemented by default for all extrinsic: Substrate PR #11431
- #[transactional] macro is deprecated: Substrate PR #11546

3.3 (HAL-03) USAGE OF ROOT ORIGIN - INFORMATIONAL

Description:

The vesting pallet is using the root origin in update_vesting_schedules function. Using such origin does not align with the blockchain paradigm of decentralization.

Code Location:

Listing 3: frame/vesting/src/lib.rs (Line 364)

```
357 #[pallet::weight(<T as Config>::WeightInfo::
 ↳ update_vesting_schedules(vesting_schedules.len() as u32))]
358 pub fn update_vesting_schedules(
359     origin: OriginFor<T>,
360     who: <T::Lookup as StaticLookup>::Source,
361     asset: AssetIdOf<T>,
362     vesting_schedules: Vec<VestingScheduleOf<T>>,
363 ) -> DispatchResult {
364     ensure_root(origin)?;
365
366     let account = T::Lookup::lookup(who)?;
367     Self::do_update_vesting_schedules(&account, asset,
 ↳ vesting_schedules)?;
368
369     Self::deposit_event(Event::VestingSchedulesUpdated { who:
 ↳ account });
370     Ok(())
371 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to not use a root origin and introduce an appropriate committee for handling administrative operations.

3.4 (HAL-04) USAGE OF SUDO PALLET - INFORMATIONAL

Description:

It was observed that the sudo pallet is part of the project. This pallet is responsible for temporarily elevating user's privileges, which does not align with the blockchain decentralization paradigm.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to remove the sudo pallet so that no user can elevate its privileges. Appropriate committees should be introduced to perform administrative operations instead.

AUTOMATED TESTING

4.1 CARGO AUDIT

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: lz4-sys
Version: 1.9.3
Title: Memory corruption in liblz4
Date: 2022-08-25
ID: RUSTSEC-2022-0051
URL: https://rustsec.org/advisories/RUSTSEC-2022-0051
Solution: Upgrade to >=1.9.4

Crate: owning_ref
Version: 0.4.1
Title: Multiple soundness issues in owning_ref
Date: 2022-01-26
ID: RUSTSEC-2022-0040
URL: https://rustsec.org/advisories/RUSTSEC-2022-0040
Solution: No safe upgrade is available!

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: 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: 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

Crate: pest
Version: 2.2.1
Warning: yanked

Crate: pest_derive
Version: 2.2.1
Warning: yanked

Crate: pest_generator
Version: 2.2.1
Warning: yanked

Crate: pest_meta
Version: 2.2.1
Warning: yanked

Crate: plotters
Version: 0.3.2
Warning: yanked
```

```
Crate: rustix
Version: 0.35.8
Warning: yanked

Crate: sp-version
Version: 5.0.0
Warning: yanked
```

THANK YOU FOR CHOOSING

// HALBORN