// HALBORN

DRAFT

Composable.Finance - Frame

Substrate Pallet Security Audit

Prepared by: **Halborn**

Date of Engagement: **August 8th, 2022 - August 26th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/26/2022	Thiago Mathias
0.2	Draft Review	08/28/2022	Timur Guvenkaya
0.3	Draft Review	08/29/2022	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
Thiago Mathias	Halborn	Thiago.Mathias@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

ComposableFinance engaged Halborn to conduct a security audit on their Substrate pallets beginning on August 8th, 2022 and ending on August 26, 2022. The security assessment was scoped to the pallets provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided three 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 pallets' functions operate as intended
- Identify potential security issues with the pallets

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which should be addressed by Composable.Finance . The main ones are the following:

- Improve proposal blacklist functionality.
- Validate the encoded proposal size before performing manipulations.
- Implement a validation routine on the metadata information.

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 frame directory in the ComposableFi/composable repository.

Pallets:

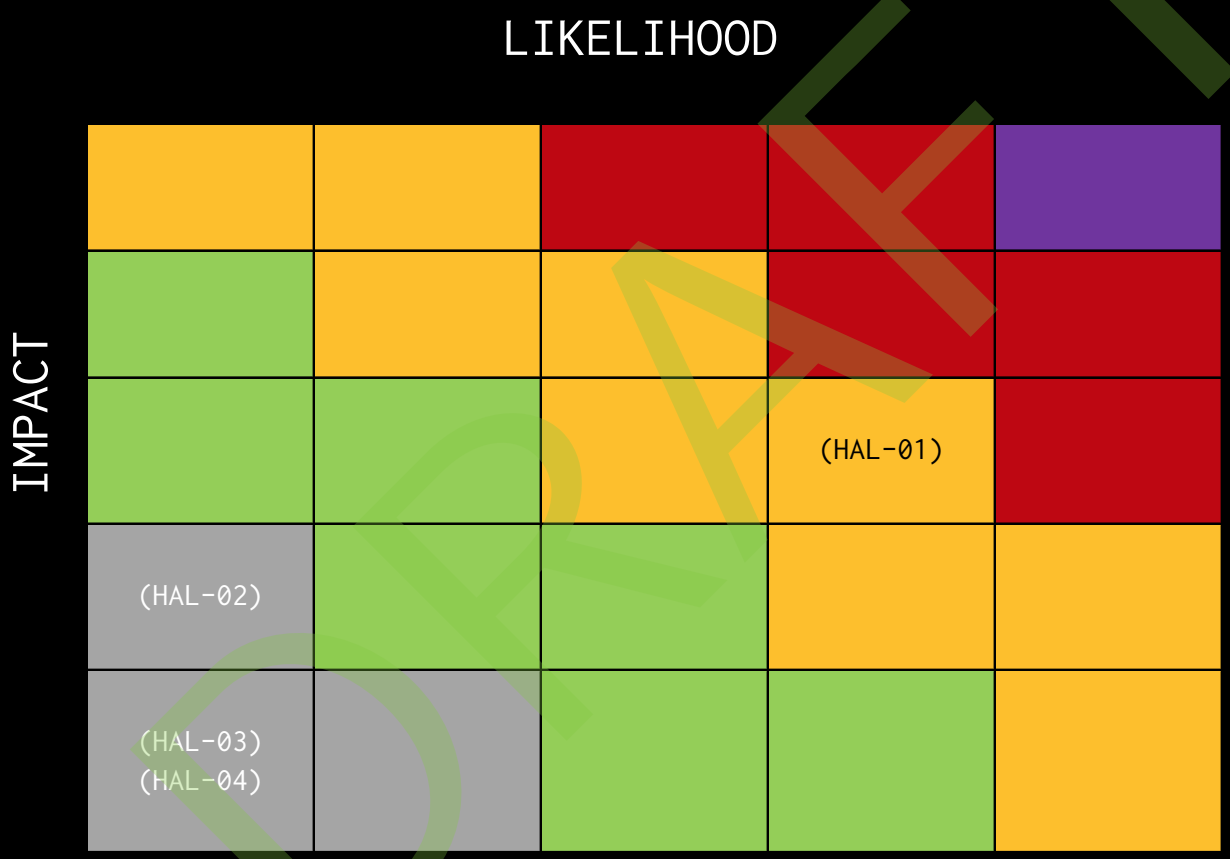
- democracy
- currency-factory
- crowdloan-rewards

Commit IDs used for the engagement:

- 491eb3e9b50b6314d3dd7964821f37fcb6f3b8c8
- 1d44536c0c316be7b09987d86f58694ecd1f5a94
- b50508d0cd98a4696103cab28fa1f9d4c8f2c7d4

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	0	3



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
POSSIBILITY OF BYPASSING THE DEMOCRACY BLACKLIST MECHANISM	Medium	-
POTENTIAL UNEXPECTED BEHAVIOUR CAUSED BY BIG PREIMAGE	Informational	-
LACK OF SANITIZATION IN FACTORY METADATA	Informational	-
USAGE OF A DEPENDENCY WITH A DEPRECATED MACRO	Informational	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) POSSIBILITY OF BYPASSING THE DEMOCRACY BLACKLIST MECHANISM - MEDIUM

Description:

The democracy pallet implements a blacklisting functionality which can be used to prevent proposing the change. The blacklisted proposal is identified by the ProposalId structure, which contains the proposal's hash and ID of an asset it is associated with. It was determined that changing the asset ID of the proposal can be used to bypass the blacklisting functionality.

Code Location:

Listing 1: frame/democracy/src/lib.rs (Lines 738,744)

```
729 #[pallet::weight(T::WeightInfo::propose())]
730 pub fn propose(
731     origin: OriginFor<T>,
732     proposal_hash: T::Hash,
733     asset_id: T::AssetId,
734     #[pallet::compact] value: BalanceOf<T>,
735 ) -> DispatchResult {
736     let who = ensure_signed(origin)?;
737     ensure!(value >= T::MinimumDeposit::get(), Error::<T>::
 ↳ ValueLow);
738     let id = ProposalId { hash: proposal_hash, asset_id };
739     let index = Self::public_prop_count().unwrap_or(0);
740     let real_prop_count = PublicProps::<T>::decode_len().unwrap_or
 ↳ (0) as u32;
741     let max_proposals = T::MaxProposals::get();
742     ensure!(real_prop_count < max_proposals, Error::<T>::
 ↳ TooManyProposals);
743
744     if let Some((until, _)) = <Blacklist<T>>::get(&id) {
745         ensure!(
746             <frame_system::Pallet<T>>::block_number() >= until,
747             Error::<T>::ProposalBlacklisted,
```

```
748             );
749         }
750
751         T::NativeCurrency::hold(&who, value)?;
752         PublicPropCount::<T>::put(index + 1);
753         <DepositOf<T>>::insert(index, (&[&who][..], value));
754
755         <PublicProps<T>>::append((index, id, who));
756
757         Self::deposit_event(Event::<T>::Proposed(index, value));
758         Ok(())
759     }
```

Risk Level:

Likelihood - 4
Impact - 3

Recommendation:

It is recommended to consider implementing a blacklisting functionality based on the change itself, i.e., it's hash instead of the object containing the change and associated asset ID.

3.2 (HAL-02) POTENTIAL UNEXPECTED BEHAVIOUR CAUSED BY BIG PREIMAGE - INFORMATIONAL

Description:

The democracy pallet requires a preimage to be submitted prior to submitting a proposal. The preimage itself is constructed by hashing the encoded proposal and matching it with its corresponding asset ID. User can submit anything as an encoded proposal, not necessarily a valid one. User submitting the preimage via note_preimage function needs to transfer some balance, which depends on the size of the encoded_proposal. The encoded proposal is represented as a Vector of bytes in the code. When the number of tokens owed for this proposal is calculated, the length of the encoded_proposal is casted to u32 type. This will work as expected for proposals of size not exceeding 4 GB; however, it might break if the user manages to send bigger proposals. Potentially, it might lead to an invalid balance calculation, where the user wouldn't pay for every byte submitted. Furthermore, as the encoded_proposal is hashed, it might also lead to a Denial of Service condition since the hashing process will take more time for larger inputs.

Code Location:

Listing 2: frame/democracy/src/lib.rs (Lines 2080,2084)

```
2075 fn note_preimage_inner(
2076     who: T::AccountId,
2077     encoded_proposal: Vec<u8>,
2078     asset_id: T::AssetId,
2079 ) -> DispatchResult {
2080     let proposal_hash = T::Hashing::hash(&encoded_proposal[..]);
2081     let id = ProposalId { hash: proposal_hash, asset_id };
2082     ensure!(!<Preimages<T>>::contains_key(&id), Error::<T>::
  ↳ DuplicatePreimage);
2083
```

```
2084     let deposit = <BalanceOf<T>>::from(encoded_proposal.len() as
  ↳ u32)
2085         .saturating_mul(T::PreimageByteDeposit::get());
2086     T::NativeCurrency::hold(&who, deposit)?;
2087
2088     let now = <frame_system::Pallet<T>>::block_number();
2089     let a = PreimageStatus::Available {
2090         data: encoded_proposal,
2091         provider: who.clone(),
2092         deposit,
2093         since: now,
2094         expiry: None,
2095     };
2096     <Preimages<T>>::insert(id, a);
2097     Self::deposit_event(Event::<T>::PreimageNoted(proposal_hash,
  ↳ who, deposit));
2098
2099     Ok(())
2100 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to implement a verification mechanism which will first check the size of `encoded_proposal` and will not perform any further computation if that size exceeds a predefined limit.

3.3 (HAL-03) LACK OF SANITIZATION IN FACTORY METADATA - INFORMATIONAL

Description:

The set_metadata function of **currency-factory** pallet does not validate data entered in metadata variable. When the function receives the **metadata** variable, no sanitization is performed to prevent problems in the front-end of the web application with malformed data, in this way, the pallet stores the information.

Code Location:

Listing 3: frame/currency-factory/src/lib.rs

```
151         #[pallet::weight(T::WeightInfo::set_metadata())]
152         pub fn set_metadata(
153             origin: OriginFor<T>,
154             asset_id: T::AssetId,
155             metadata: BasicAssetMetadata,
156         ) -> DispatchResultWithPostInfo {
157             T::AddOrigin::ensure_origin(origin)?;
158             if AssetEd::<T>::get(asset_id).is_some() {
159                 // note: if will decide to build route on symbol,
 ↳ than better to make second map
160                 // from symbol to asset to check unique
161                 AssetMetadata::<T>::insert(asset_id, metadata);
162                 Ok(().into())
163             } else {
164                 Err(Error::<T>::AssetNotFound.into())
165             }
166         }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Implement a validation routine in the `submit_proposal` function on the **currency-factory** pallet to validate the characters entered.

3.4 (HAL-04) USAGE OF A DEPENDENCY WITH A DEPRECATED MACRO - INFORMATIONAL

Description:

The pallets are using #[transactional] macro in order to assure that the changes are committed to storage only after the whole function is completed. This macro was marked deprecated as since polkadot version 0.9.25 this is the default behaviour of the extrinsic.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to upgrade the polkadot dependency to a newer version, which implements #[transactional] macro behaviour by default for every extrinsic.

Reference:

- The behavior of #[transactional] macro is the default: Substrate PR #11431
- #[transactional] macro is deprecated: Substrate PR #11546

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

THANK YOU FOR CHOOSING

// HALBORN