Composable.Finance - BYO Gas

Substrate Pallet Security Audit

Prepared by: **Halborn**

Date of Engagement: **September 26th, 2022 - October 4th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/03/2022	Michal Bajor
0.2	Draft Review	10/03/2022	Timur Guvenkaya
0.3	Draft Review	10/03/2022	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 September 26th, 2022 and ending on October 4th, 2022. The security assessment was scoped to the Substrate pallets provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided one week 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.

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

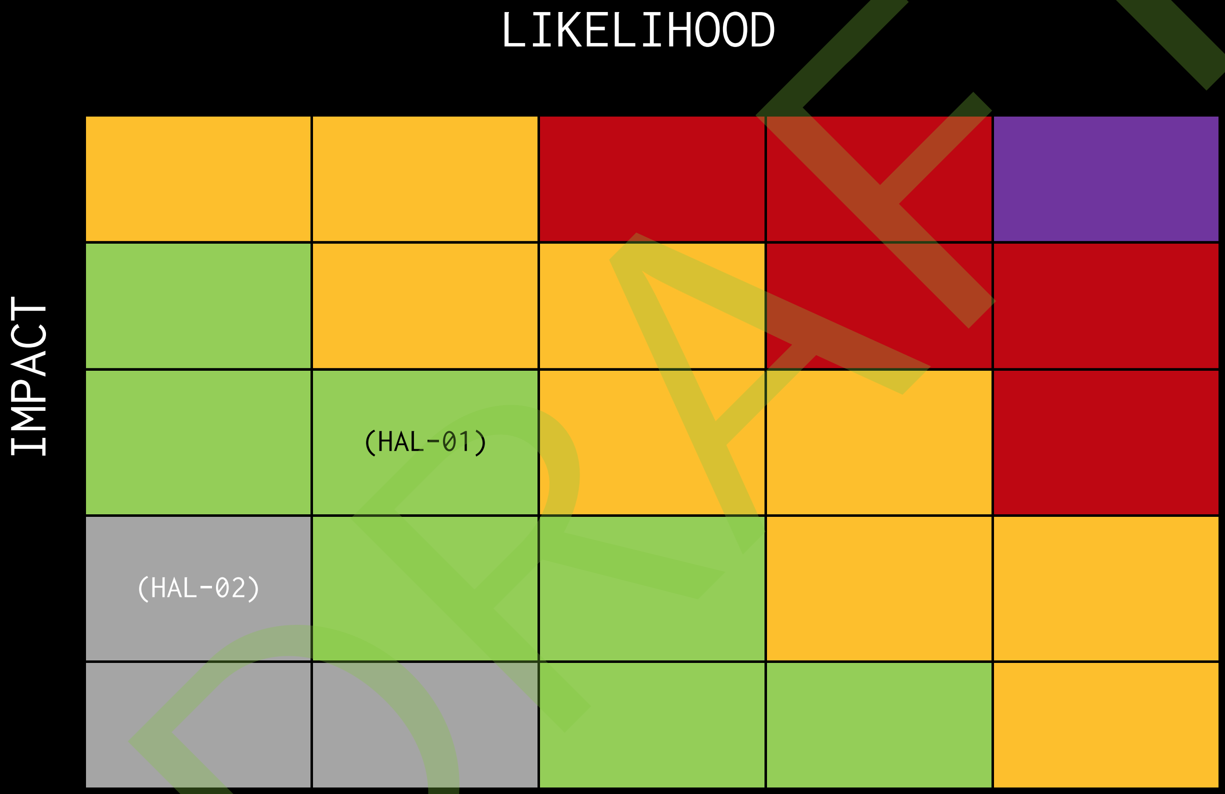
The review was scoped to the asset-tx-payment pallet in frame/transaction-payment directory in the ComposableFi/substrate repository.

Commit ID used for the engagement:

- f709a3d3b8f116f8b7e92ec56bc1ca5a0409eaa5

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	1



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
CONFIGURATION ORIGIN CAN BE USED TO SET A PAYMENT ASSET FOR AN ARBITRARILY CHOSEN USER	Low	-
PRESENCE OF TESTING CODE	Informational	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) CONFIGURATION ORIGIN CAN BE USED TO SET A PAYMENT ASSET FOR AN ARBITRARILY CHOSEN USER - LOW

Description:

The asset-tx-payment pallet defines an extrinsic set_payment_asset element that allows users to set the asset they want to use for fees associated with using the blockchain. However, the function makes sure that the asset is set for the user who called the extrinsic only if the caller is not a ConfigurationOrigin. As such, this origin can be used to arbitrarily set the asset for any given user.

Code Location:

Listing 1: substrate/frame/transaction-payment/asset-tx-payment/src/lib.rs (Line 191)

```
184 #[pallet::weight(T::WeightInfo::set_payment_asset())]
185 pub fn set_payment_asset(
186     origin: OriginFor<T>,
187     payer: T::AccountId,
188     asset_id: Option<ChargeAssetIdOf<T>>,
189 ) -> DispatchResult {
190     // either configuration origin or owner of configuration
191     if let Err(origin) = T::ConfigurationOrigin::try_origin(origin
  ↳ ) {
192         let who = ensure_signed(origin)?;
193         ensure!(who == payer, DispatchError::BadOrigin,)
194     };
195 
196     // clean previous configuration
197     if let Some((asset_id, ed)) = <PaymentAssets<T>>::get(&payer)
  ↳ {
198         T::Lock::release(asset_id, &payer, ed, true)?;
199         <PaymentAssets<T>>::remove(&payer);
200     }
```

```

    // configure new payment asset and hold some ed
    if let Some(asset_id) = asset_id {
        let ed = T::BalanceConverter::to_asset_balance(
            T::ConfigurationExistentialDeposit::get(),
            asset_id,
        )
        .map_err(|_| DispatchError::Other("Cannot convert ED to
 ↳ asset balance"))?;
        T::Lock::hold(asset_id, &payer, ed)?;
        <PaymentAssets<T>>::insert(payer, (asset_id, ed));
    }

    Ok(())
}
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It is recommended not to allow any type of bypass in the configuration of the payment asset.

3.2 (HAL-02) PRESENCE OF TESTING CODE - INFORMATIONAL

Description:

The asset-tx-payment defines a ChargeAssetTxPayment structure, the implementation of which is responsible for calculating the fees associated with calls. The impl block defines a from function that allows you to manually set the self.asset_id variable. This is the asset (if configured) that will be returned by the get_payment_asset function used in the validate and pre_dispatch functions. The from function is used for testing and should not be present in the production environment.

Code Location:

The from function:

Listing 2: substrate/frame/transaction-payment/asset-tx-payment/src/lib.rs (Line 240)

```rust
pub fn from(tip: BalanceOf<T>, asset_id: Option<ChargeAssetIdOf<T
↳ >>) -> Self {
    Self { tip, asset_id }
}
```

The get_payment asset function:

Listing 3: substrate/frame/transaction-payment/asset-tx-payment/src/lib.rs (Line 284)

```rust
fn get_payment_asset(&self, who: &T::AccountId, call: &T::Call) ->
↳ Option<ChargeAssetIdOf<T>> {
    if self.asset_id.is_some() || !<T as Config>::
↳ UseUserConfiguration::get() {
        return self.asset_id
    }

    let call = <T as Config>::PayableCall::from_ref(call);
    match call.is_sub_type() {
        Some(Call::set_payment_asset { asset_id, .. }) => asset_id.to_owned(),
        _ => <PaymentAssets<T>>::get(who).map(|x| x.0),
    }
}
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended not to implement functions to production builds that are used for testing purposes.

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: lz4-sys
Version: 1.9.2
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: 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: 0.1.2
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:  memmap
Version:  0.7.0
Warning:  unmaintained
Title:  memmap is unmaintained
Date:  2020-12-02
ID: RUSTSEC-2020-0077
URL: https://rustsec.org/advisories/RUSTSEC-2020-0077

Crate:  serde_cbor
Version:  0.11.1
Warning:  unmaintained
Title:  serde_cbor is unmaintained
Date:  2021-08-15
ID: RUSTSEC-2021-0127
URL: https://rustsec.org/advisories/RUSTSEC-2021-0127

Crate:  blake2
Version:  0.10.2
Warning:  yanked

Crate:  block-buffer
Version:  0.10.0
Warning:  yanked

Crate:  cpufeatures
Version:  0.2.1
Warning:  yanked

Crate:  sha2
Version:  0.9.8
Warning:  yanked

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

THANK YOU FOR CHOOSING

// HALBORN