// HALBORN

Composable - IBC Pallet

Substrate Pallet Security Audit

Prepared by: **Halborn**

Date of Engagement: **October 10th, 2022 - November 4th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/10/2022	Thiago Mathias
0.2	Draft Review	11/04/2022	Timur Guvenkaya
0.3	Draft Review	11/17/2022	Gabi Urrutia
1.0	Remediation Plan	11/24/2022	Michal Bajor
1.1	Remediation Plan Review	11/24/2022	Timur Guvenkaya
1.2	Remediation Plan Review	11/24/2022	Piotr Cielas
1.3	Remediation Plan Review	11/24/2022	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
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com
Timur Guvenkaya	Halborn	Timur.Guvenkaya@halborn.com
Thiago Mathias	Halborn	Thiago.Mathias@halborn.com
Michal Bajor	Halborn	Michal.Bajor@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Composable engaged Halborn to conduct a security audit on their IBC Pallet beginning on October 10th, 2022 and ending on November 4th, 2022. The security assessment was scoped to the pallet provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided four weeks for the engagement and assigned two full-time security engineers to audit the security of the IBC pallet. Security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

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

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

- Remove deprecated macros
- Validate amount to be transferred.

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 IBC 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.
- On chain testing of core functions(polkadot.js).
- 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

Code repositories:

1. Substrate Composable pallets
   (a) Repository: ComposableFi/composable
   (b) Commit ID: cdce3144be13626623d561292d4ee6a54ffaf498
   (c) Pallets in scope:
      - IBC (composable/code/parachain/frame/ibc)

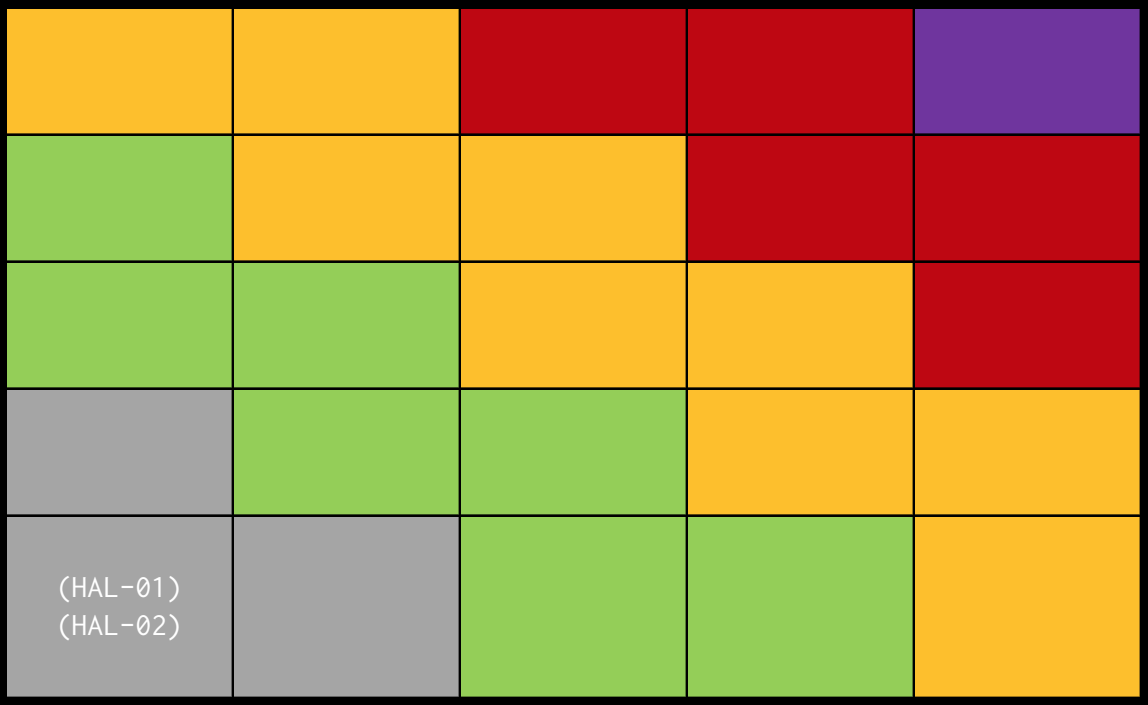
Out-of-scope: External libraries and financial related attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	2

LIKELIHOOD

IMPACT



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
ALLOWED TO TRANSFER ZERO AMOUNT	Informational	ACKNOWLEDGED
USAGE OF DEPRECATED MACRO	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) ALLOWED TO TRANSFER ZERO AMOUNT - INFORMATIONAL

Description:

The **ibc** pallet defines a transfer function that allows users to transfer zero amounts. This condition can be abused if someone constantly calls transfer with zero amount and thus filling up the block space.

Code Location:

Listing 1: ibc/src/lib.rs (Lines 613,685)

```
574 #[frame_support::transactional]
575 #[pallet::weight(<T as Config>::WeightInfo::transfer())]
576 pub fn transfer(
577     origin: OriginFor<T>,
578     params: TransferParams<T::AccountId>,
579     asset_id: <T as DeFiComposableConfig>::MayBeAssetId,
580     amount: <T as DeFiComposableConfig>::Balance,
581 ) -> DispatchResult {
582     let origin = ensure_signed(origin)?;
583     // Check if it's a local asset id, native asset or an ibc
 ↳ asset id
584     // If native or local asset, get the string representation of
 ↳ the asset
585     let denom = if let Some(denom) = IbcAssetIds::<T>::get(
 ↳ asset_id) {
586         String::from_utf8(denom).map_err(|_| Error::<T>::Utf8Error
 ↳ )?
587     } else {
588         let asset_id: CurrencyId = asset_id.into();
589         CurrencyId::native_asset_name(asset_id.0)
590             .map(|val| val.to_string())
591             .unwrap_or_else(|_| asset_id.to_string())
592     };
593
594     let account_id_32: AccountId32 = origin.clone().into();
595     let from = runtime_interface::account_id_to_ss58(account_id_32
 ↳ .into())
596         .and_then(|val| {
```

```
597             String::from_utf8(val).map_err(|_| SS58CodecError::
 ↳ InvalidAccountId)
598         })
599         .map_err(|_| Error::<T>::Utf8Error)?;
600     let to = match params.to {
601         MultiAddress::Id(id) => {
602             let account_id_32: AccountId32 = id.into();
603             runtime_interface::account_id_to_ss58(account_id_32.
 ↳ into())
604                 .and_then(|val| {
605                     String::from_utf8(val).map_err(|_|
 ↳ SS58CodecError::InvalidAccountId)
606                 })
607                 .map_err(|_| Error::<T>::Utf8Error)?
608         },
609         MultiAddress::Raw(bytes) =>
610             String::from_utf8(bytes).map_err(|_| Error::<T>::
 ↳ Utf8Error)?,
611     };
612     let denom = PrefixedDenom::from_str(&denom).map_err(|_| Error
 ↳ ::<T>::InvalidIbcDenom)?;
613     let ibc_amount = Amount::from_str(&format!("{:?}", amount))
614         .map_err(|_| Error::<T>::InvalidAmount)?;
615     let coin = PrefixedCoin { denom, amount: ibc_amount };
616     let source_channel = ChannelId::new(params.source_channel);
617     let source_port = PortId::transfer();
618     let (latest_height, latest_timestamp) =
619         Pallet::<T>::latest_height_and_timestamp(&source_port, &
 ↳ source_channel)
620             .map_err(|_| Error::<T>::TimestampAndHeightNotFound)?;
621
622     let (timeout_height, timeout_timestamp) = match params.timeout
 ↳  {
623         Timeout::Offset { timestamp, height } => {
624             let timestamp = timestamp
625                 .map(|offset| (latest_timestamp + Duration::
 ↳ from_secs(offset)))
626                 .transpose()
627                 .map_err(|_| Error::<T>::InvalidTimestamp)?
628                 .unwrap_or_default();
629             let height = height.map(|offset| latest_height.add(
 ↳ offset)).unwrap_or_default();
630             (height, timestamp)
631         },
```

```
632         Timeout::Absolute { timestamp, height } => {
633             let timestamp = timestamp
634                 .map(Timestamp::from_nanoseconds)
635                 .transpose()
636                 .map_err(|_| Error::<T>::InvalidTimestamp)?
637                 .unwrap_or_default();
638             let height = height
639                 .map(|revision_height| {
640                     Height::new(latest_height.revision_number,
 ↳ revision_height)
641                 })
642                 .unwrap_or_default();
643             (height, timestamp)
644         },
645     };
646
647     let msg = MsgTransfer {
648         source_port,
649         source_channel,
650         token: coin.clone(),
651         sender: Signer::from_str(&from).map_err(|_| Error::<T>::
 ↳ Utf8Error)?,
652         receiver: Signer::from_str(&to).map_err(|_| Error::<T>::
 ↳ Utf8Error)?,
653         timeout_height,
654         timeout_timestamp,
655     };
656
657     if is_sender_chain_source(msg.source_port.clone(), msg.
 ↳ source_channel, &msg.token.denom)
658     {
659         // Store escrow address, so we can use this to identify
 ↳ accounts to keep alive when
660         // making transfers in callbacks Escrow addresses do not
 ↳ need to be kept alive
661         let escrow_address =
662             get_channel_escrow_address(&msg.source_port, msg.
 ↳ source_channel)
663                 .map_err(|_| Error::<T>::ChannelEscrowAddress)?;
664         let account_id = T::AccountIdConversion::try_from(
 ↳ escrow_address)
665             .map_err(|_| Error::<T>::ChannelEscrowAddress)?
666             .into_account();
```

```
667         let _ = EscrowAddresses::<T>::try_mutate::<_, &'static str
 ↳ , _>(|addresses| {
668             if !addresses.contains(&account_id) {
669                 addresses.insert(account_id);
670                 Ok(())
671             } else {
672                 Err("Address already exists")
673             }
674         });
675     }
676
677     Pallet::<T>::send_transfer(msg).map_err(|e| {
678         log::trace!(target: "pallet_ibc", "[transfer]: error: {:?}
 ↳ ", e);
679         Error::<T>::TransferFailed
680     })?;
681
682     Self::deposit_event(Event::<T>::TokenTransferInitiated {
683         from: origin,
684         to: to.as_bytes().to_vec(),
685         amount,
686         local_asset_id: Pallet::<T>::ibc_denom_to_asset_id(
687             coin.denom.to_string(),
688             coin.clone(),
689         ),
690         ibc_denom: coin.denom.to_string().as_bytes().to_vec(),
691     });
692     Ok(())
693 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

It is recommended to add checks to ensure the amount to be transferred is bigger than 0.

Remediation Plan:

ACKNOWLEDGED: The Composable Finance team acknowledged this issue.

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

Description:

The IBC pallet is using #[transactional] macro to add another storage layer to function execution, making it not to alter storage if an error is encountered. However, the IBC pallet is using Polkadot modules in version 0.9.27 which implements this behavior by default for all functions.

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

It is recommended to delete the unnecessary macro.

Remediation Plan:

ACKNOWLEDGED: The Composable Finance team acknowledged this issue.

THANK YOU FOR CHOOSING

// HALBORN