Manta Network - SBT Module

Substrate Pallet Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 25th, 2023 - May 26th, 2023**

Visit: **Halborn.com**

DRAFT

DRAFT

DRAFT

DRAFT

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/28/2023	Gonzalo Junquera
0.2	Document Updates	05/05/2023	Gonzalo Junquera
0.3	Document Updates	05/12/2023	Gonzalo Junquera
0.4	Document Updates	05/19/2023	Gonzalo Junquera
0.5	Document Updates	05/28/2023	Gonzalo Junquera
0.6	Draft Version	05/29/2023	Gonzalo Junquera
0.7	Draft Review	05/29/2023	Alp Onaran
0.8	Draft Review	05/29/2023	Piotr Cielas
0.9	Draft Review	05/29/2023	Gabi Urrutia

DRAFT

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
Alp Onaran	Halborn	Alpcan.Onaran@halborn.com
Gonzalo Junquera	Halborn	Gonzalo.Junquera@halborn.com

DRAFT

EXECUTIVE OVERVIEW

DRAFT

1.1 INTRODUCTION

Manta Network is committed to enhancing privacy for Web 3.0 applications via the application of zk-SNARKs - zero-knowledge proof cryptographic schemes. These schemes allow proving ownership of particular information without exposing the information itself.

Manta Network , functioning as a ZK Layer 1 blockchain, aims to establish a fast and decentralized framework for the development of privacy-focused applications within the Web3 ecosystem.

To evaluate the security of their new protocol for minting zkSBT tokens, the Manta Network engaged Halborn. The audit began on April 25th, 2023 and concluded on May 26th, 2023 . More details on the scope of this audit, including commit hashes, are available in the Scope section of this report. The principal pallet undergoing review was ‘manta-sbt’, which serves as the entry point for creating non-transferable, soul-bound NFTs as unspendable UTXOs.

1.2 AUDIT SUMMARY

The Halborn team was allocated five weeks for the engagement, during which a full-time security engineer was assigned to audit the security of the pallets in scope, along with configurations and changes in the primitives, node and runtime folders. The engineer is an expert in blockchain and smart contract security, boasting advanced penetration testing and smart-contract hacking skills, as well as a deep understanding of multiple blockchain protocols.

The objectives of this audit include:

- Verification of whether the zkSBT can be transferred.
- Confirmation of the uniqueness of the minted zkSBT.
- Investigation of potential security vulnerabilities that could lead to a Denial of Service (DoS), logical errors, abnormal behavior or

DRAFT

malicious actions.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the pallet audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of pallets 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).
- Fuzz of private and public functions
- Scanning dependencies for known vulnerabilities (cargo audit).

DRAFT

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

DRAFT

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

DRAFT

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

DRAFT

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

DRAFT

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

DRAFT

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

DRAFT

2.4 SCOPE

Code repository:

1. Manta

- Repository: Manta
- Commit ID: ceb9e46cd53b77eb914ba6c17452fc238bc3a28f
- Pallets in scope:

    1. SBT pallet
    2. Support pallet

- Directories in scope:

    1. node/*
    2. runtime/*
    3. primitives/*

DRAFT

2.5 APPLICATION FLOW ANALYSIS

This section of the report provides an in-depth analysis of the application flow, specifically focusing on the process of minting Zero-Knowledge Soulbound Tokens (zkSBT). These tokens represent an advancement of traditional Soulbound tokens, leveraging Zero-Knowledge Proofs (ZKPs) for enhanced security and privacy. They are not only straightforward to mint, but also easy to utilize for verification purposes. Notably, these tokens are non-transferable. The metadata associated with a zkSBT is as adaptable as that of any conventional SBT, accommodating various data types such as a Profile Picture (PFP), conventional or AI-generated photos, social graphs, as well as Web 2.0 handles including Twitter and Instagram.

Actors:

This section defines various roles within the system and outlines the functionalities associated with each:

- Users
  This group includes individuals or organizations interacting with the protocol. Users can mint zkSBT by paying with KMA/MANTA or by having an authorized EVM address.
- Admin Custom Origin: Administrators are responsible for creating an account that can add Ethereum Virtual Machine (EVM) addresses to an approved whitelist. They also have the authority to enable or disable the minting of zkSBT through EVM addresses.
- AllowList Account: This is the account responsible for adding EVM addresses to the whitelist, thus granting them the ability to mint one zkSBT.

The functions specific to each role are as follows:

- User Functions:

DRAFT

1. reserve_sbt: Reserves AssetIds for future token minting. Each zkSBT possesses a unique id.
2. to_private: Mints a zkSBT, but the user must first reserve some ids.
3. mint_sbt_eth: Mints a zkSBT using a signature from an EVM address.

- Admin custom Origin Functions:

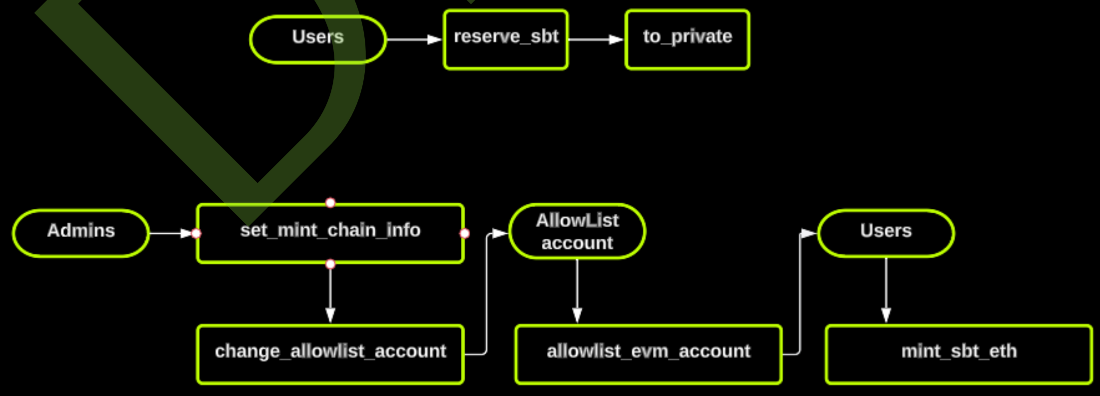
1. change_allowlist_account: Assigns the privileged AllowList Account.
2. set_mint_chain_info: Sets the specific time range during which token minting will be possible for a particular chain.

- AllowList account:

1. allowlist_evm_account: Adds an EvmAddress to the approved addresses list and reserves a unique AssetId for the account.

Workflows:

The code facilitates two distinct methods for users to mint zkSBT tokens. The first method involves using the native token (KMA/MANTA) to reserve the right to mint, followed by the actual minting of the zkSBT. The alternative method allows a user to mint one free zkSBT if their 'EvmAddress' has been added to an allowlist.



Users must initially reserve some IDs in order to mint zkSBT. These IDs are reserved by paying the necessary amount with native tokens, after which users can continue minting zkSBT until they exhaust their reserved IDs.

Alternatively, users can mint one free zkSBT if their EVM address has been whitelisted and the minting process is currently available (i.e., the present time falls within the minting interval).

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	4	0

DRAFT

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LOSS OF RESERVED SBT IDS	Low (2.5)	-
LAST SBT IDS CANNOT BE RESERVED	Low (2.5)	-
DOWNCASTING OF 64-BIT INTEGER	Low (2.5)	-
UNCHECKED MATH COULD IMPACT WEIGHT CALCULATION	Low (2.5)	-

DRAFT

FINDINGS & TECH DETAILS

DRAFT

4.1 (HAL-01) LOSS OF RESERVED SBT IDS - LOW (2.5)

Description:

The reserve_sbt function calculates a range of IDs and stores this range in the ReservedIds storage map, using the caller's address as the key. It was identified that users lose their reserved SBT IDs when they call the reserve_sbt function without first minting their previously reserved SBT IDs. This occurs because the previous reserved range is overwritten.

Code Location:

Body of the reserve_sbt function:

Listing 1: pallets/manta-sbt/src/lib.rs (Line 376)

```
368 let asset_id_range: Vec<StandardAssetId> = (0..T::MintsPerReserve
    ↳ ::get())
369                 .map(|_| Self::next_sbt_id_and_increment())
370                 .collect::<Result<Vec<StandardAssetId>, _>>()?;
371
372             // The range of `AssetIds` that are reserved as SBTs
373             let start_id: StandardAssetId = *asset_id_range.first
    ↳ ().ok_or(Error::<T>::ZeroMints)?;
374             let stop_id: StandardAssetId = *asset_id_range.last().
    ↳ ok_or(Error::<T>::ZeroMints)?;
375
376             ReservedIds::<T>::insert(&who, (start_id, stop_id));
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (2.5)

DRAFT

Proof Of Concept:

This test reserves ids two times and mints one zkSBT. The first zkSBT token will not have the id 1, it will have the id 6 instead.

Listing 2: pallets/manta-sbt/src/tests.rs

```rust
#[test]
fn hal01() {
    let mut rng = OsRng;
    new_test_ext().execute_with(|| {
        assert_ok!(Balances::set_balance(
            MockOrigin::root(),
            ALICE,
            1_000_000_000_000_000,
            0
        ));
        //Reserve IDs from 1 to 5
        assert_ok!(MantaSBTPallet::reserve_sbt(MockOrigin::signed(ALICE)));
        //Reserve IDs from 6 to 10
        assert_ok!(MantaSBTPallet::reserve_sbt(MockOrigin::signed(ALICE)));

        let value = 1;
        let id = field_from_id(ReservedIds::<Test>::get(ALICE).unwrap().0);
        let post = sample_to_private(id, value, &mut rng);
        assert_ok!(MantaSBTPallet::to_private(
            MockOrigin::signed(ALICE),
            Box::new(post),
            bvec![0]
        ));

        //The first zkSBT minted has the id 6.
        assert_eq!(
            SbtMetadata::<Test>::get(6).unwrap().extra,
            Some(bvec![0])
        );
    });
}
```

DRAFT

Recommendation:

To resolve this issue, it is recommended to restrict users from reserving additional SBT IDs if they have not minted their previously reserved IDs.

DRAFT

4.2 (HAL-02) LAST SBT IDS CANNOT BE RESERVED - LOW (2.5)

DRAFT

Description:

When users invoke the reserve_sbt function, it reserves a specific number of IDs - quantified by MintPerReserve. The reserve_sbt function achieves this by repeatedly calling the next_sbt_id_and_increment function - as many times as the MintPerReserve value. This next_sbt_id_and_increment function serves to return the next available ID and concurrently increment the NextSbtId storage value by 1.

A potential problem arises if the incrementing process results in an overflow, causing the next_sbt_id_and_increment function to throw an overflow exception, which in turn fails the ongoing transaction. In this scenario, previously identified IDs that did not contribute to the overflow situation remain unreserved. This issue presents a concern as it could potentially lead to resource allocation inefficiencies and transaction failures.

Code Location:

Body of the reserve_sbt function, where the next zkSBT id is incremented.

Listing 3: pallets/manta-sbt/src/lib.rs (Line 369)

```
pub fn reserve_sbt(origin: OriginFor<T>) -> DispatchResult {
    let who = ensure_signed(origin)?;

    // Charges fee to reserve AssetIds
    <T as pallet::Config>::Currency::transfer(
        &who,
        &Self::account_id(),
        T::ReservePrice::get(),
        ExistenceRequirement::KeepAlive,
    )?;

    // Reserves uniques AssetIds to be used later to mint SBTs
```

```
    let asset_id_range: Vec<StandardAssetId> = (0..T::MintsPerReserve::get())
        .map(|_| Self::next_sbt_id_and_increment())
        .collect::<Result<Vec<StandardAssetId>, _>>()?;
```

next_sbt_id_and_increment function will overflow if the max number for u128 is surpassed

Listing 4: pallets/manta-sbt/src/lib.rs (Line 883)

```
875 fn next_sbt_id_and_increment() -> Result<StandardAssetId, DispatchError> {
876         NextSbtId::<T>::try_mutate(|maybe_val| {
877             match maybe_val {
878                 Some(current) => {
879                     let id = *current;
880                     *maybe_val = Some(
881                         current
882                             .checked_add(One::one())
883                             .ok_or(ArithmeticError::Overflow)?,
884                     );
885                     Ok(id)
886                 }
887                 // If storage is empty, starts at value of one (Field cannot be zero)
888                 None => {
889                     *maybe_val = Some(2);
890                     Ok(One::one())
891                 }
892             }
893         })
894     }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:U (2.5)

DRAFT

Proof Of Concept:

Note: For this Proof of Concept (PoC), the codebase was modified such that the zkSBT IDs are now u8 instead of u128. This alteration reduces the time needed to demonstrate that the function fails in this edge-case scenario.

In this test, we reserve all available IDs, excluding the last five. Attempting to reserve the last ID will cause the StandardAssetId value to overflow, resulting in a failure.

Listing 5: pallets/manta-sbt/src/tests.rs

```rust
#[test]
fn hal02() {
    new_test_ext().execute_with(|| {
        assert_ok!(Balances::set_balance(
            MockOrigin::root(),
            ALICE,
            1_000_000_000_000_000,
            0
        ));
        for i in (1..51) {
            assert_ok!(MantaSBTPallet::reserve_sbt_bis(MockOrigin::signed(ALICE)));
            println!("First id: {} - Last id: {}", ReservedIdsBis::<Test>::get(ALICE).unwrap().0,ReservedIdsBis::<Test>::get(ALICE).unwrap().1);
        }

        assert_noop!(MantaSBTPallet::reserve_sbt_bis(MockOrigin::signed(ALICE)),ArithmeticError::Overflow);
    });
}
```

DRAFT

Recommendation:

To address this issue, it is recommended to implement a check to determine whether the value of the StandardAssetId has reached the maximum value for u128 can prevent overflow. This measure will stop the occurrence of an exception.

4.3 (HAL-03) DOWNCASTING OF 64-BIT INTEGER - LOW (2.5)

DRAFT

Description:

It was observed that in certain circumstances, usize values are cast to types such as u8 and u32. The usize data type in the Rust programming language represents a pointer-sized unsigned integer. The actual size of usize is dependent on the platform: it's 32 bits on a 32-bit platform and 64 bits on a 64-bit platform. Consequently, depending on the system, there could be a cast from an u64 to an u32. This implies that an attempt could be made to store a value larger than the maximum value that can be held in an u32, leading to unexpected consequences.

Code Location:

Usize is casted to u8:

Listing 6: pallets/manta-sbt/src/lib.rs (Line 783)

```
768 fn pull_receivers(
769         receiver_indices: [usize; MerkleTreeConfiguration::
  ↳ FOREST_WIDTH],
770         max_update_request: u64,
771     ) -> (bool, ReceiverChunk) {
772         let mut more_receivers = false;
773         let mut receivers = Vec::new();
774         let mut receivers_pulled: u64 = 0;
775         let max_update = if max_update_request > Self::
  ↳ PULL_MAX_RECEIVER_UPDATE_SIZE {
776             Self::PULL_MAX_RECEIVER_UPDATE_SIZE
777         } else {
778             max_update_request
779         };
780
781         for (shard_index, utxo_index) in receiver_indices.into_iter
  ↳ ().enumerate() {
782             more_receivers |= Self::pull_receivers_for_shard(
783                 shard_index as u8,
784                 utxo_index,
```

```
785                 max_update,
786                 &mut receivers,
787                 &mut receivers_pulled,
788             );
789             // NOTE: If max capacity is reached and there is more
 ↳ to pull, then we return.
790             if receivers_pulled == max_update && more_receivers {
791                 break;
792             }
793         }
794         (more_receivers, receivers)
795     }
```

Usize is casted to u32:

Listing 7: pallets/manta-support/src/manta_pay.rs (Line 860)

```
867 impl TryFrom<merkle_tree::CurrentPath<MerkleTreeConfiguration>>
 ↳ for CurrentPath {
868     type Error = Error;
869 
870     #[inline]
871     fn try_from(path: merkle_tree::CurrentPath<
 ↳ MerkleTreeConfiguration>) -> Result<Self, Error> {
872         Ok(Self {
873             sibling_digest: fp_encode(path.sibling_digest)?,
874             leaf_index: path.inner_path.leaf_index.0 as u32,
875             inner_path: path
876                 .inner_path
877                 .path
878                 .into_iter()
879                 .map(fp_encode)
880                 .collect::<Result<_, _>>()?,
881         })
882     }
883 }
```

Listing 8: pallets/manta-support/src/manta_pay.rs (Lines 1095,1096)

```
1091 impl From<RawCheckpoint> for Checkpoint {
1092     #[inline]
1093     fn from(checkpoint: RawCheckpoint) -> Self {
```

DRAFT

```rust
        Self::new(
            checkpoint.receiver_index.map(|i| i as usize).into(),
            checkpoint.sender_index as usize,
        )
    }
}
```

Listing 9: runtime/calamari/src/migrations/staking.rs (Line 70)

```rust
70 let n_of_candidates = manta_collator_selection::Pallet::<T>::
   ↳ candidates().len() as u32;
71 let new_n_of_candidates = n_of_candidates + invulnerables.len() as
   ↳  u32;
```

Recommendation:

To address this issue, it is recommended to check the value against the maximum value before casting.

DRAFT

4.4 (HAL-04) UNCHECKED MATH COULD IMPACT WEIGHT CALCULATION - LOW (2.5)

Description:

It was identified that several areas in the buy_weight and the refund_weight functions that could potentially benefit from enhanced computational checks. Currently, despite numerous instances of proven arithmetic calculations, the function does not have a mechanism to handle situations where underflow or overflow states might occur. While these states haven't been identified as potential risks for exploitation, implementing additional safeguards to account for them will be beneficial.

Another point of consideration pertains to the WEIGHT_PER_SECOND value. This value serves as a divisor in computing the number of tokens required for payment or refund during the weight purchasing procedure. While it is predetermined as a constant during the system's compilation, it currently lacks a constraint to assure that it never equals zero. This is a significant potential risk as it could result in a system panic if the value happens to be zero, causing a division by zero error. Moreover, as the WEIGHT_PER_SECOND value is also used in calculations elsewhere in the system, this issue could potentially affect other sections of the codebase as well.

DRAFT

Code Location:

Unsafe multiplication in the tests

multiplier_growth_simulator_and_congestion_budget_test:

Listing 10: runtime/calamari/src/fee.rs (Line 76)

```
#[test]
    #[ignore] // This test should not fail CI
```

```
fn multiplier_growth_simulator_and_congestion_budget_test() {
    let target_daily_congestion_cost_usd = 100_000;
    let kma_price = fetch_kma_price().unwrap();
    println!("KMA/USD price as read from CoinGecko = {kma_price}");
    let target_daily_congestion_cost_kma =
        (target_daily_congestion_cost_usd as f32 / kma_price * KMA as f32) as u128;

```

Unsafe multiplication in buy_weight function

Listing 11: primitives/manta/src/xcm.rs (Line 183)

```
fn buy_weight(&mut self, weight: Weight, payment: Assets) -> Result<Assets> {
    log::debug!(
        target: "FirstAssetTrader::buy_weight",
        "weight: {:?}, payment: {:?}",
        weight,
        payment
    );

    let first_asset = payment.fungible_assets_iter().next().ok_or({
        log::debug!(
            target: "FirstAssetTrader::buy_weight",
            "no assets in payment: {:?}",
            payment,
        );
        XcmError::TooExpensive
    })?;

    // Check the first asset
    match (first_asset.id, first_asset.fun) {
        (XcmAssetId::Concrete(id), Fungibility::Fungible(_)) => {
            let asset_id = M::asset_id(&id.clone().into()).ok_or({
                log::debug!(
                    target: "FirstAssetTrader::buy_weight",
                    "asset_id missing for asset location with id: {:?}",
                    id,
                );
```

DRAFT

```
                XcmError::TooExpensive
            })?;
            let units_per_second = M::units_per_second(&asset_id).ok_or({
                log::debug!(
                    target: "FirstAssetTrader::buy_weight",
                    "units_per_second missing for asset with id: {:?}",
                    id,
                );
                XcmError::TooExpensive
            })?;

            let amount = units_per_second * (weight as u128) / (WEIGHT_PER_SECOND as u128);
            // we don't need to proceed if amount is zero.
            // This is very useful in tests.
            if amount.is_zero() {
                return Ok(payment);
            }
```

Unsafe substraction in refund_weight function

Listing 12: primitives/manta/src/xcm.rs (Line 251)

```
248 fn refund_weight(&mut self, weight: Weight) -> Option<MultiAsset> {
249         if let Some((id, prev_amount, units_per_second)) = &mut self.refund_cache {
250             let weight = weight.min(self.weight);
251             self.weight -= weight;
252             let amount = *units_per_second * (weight as u128) / (WEIGHT_PER_SECOND as u128);
253             *prev_amount = prev_amount.saturating_sub(amount);
254             Some(MultiAsset {
255                 fun: Fungibility::Fungible(amount),
256                 id: XcmAssetId::Concrete(id.clone()),
257             })
258         } else {
259             None
260         }
261     }
```

DRAFT

Places where `WEIGHT_PER_SECOND` is used as a divisor:

- Function `refund_weight`

Listing 13: primitives/manta/src/xcm.rs (Line 252)

```
247 #[inline]
248 fn refund_weight(&mut self, weight: Weight) -> Option<MultiAsset>
  ↳ {
249     if let Some((id, prev_amount, units_per_second)) = &mut self.
  ↳ refund_cache {
250         let weight = weight.min(self.weight);
251         self.weight -= weight;
252         let amount = *units_per_second * (weight as u128) / (
  ↳ WEIGHT_PER_SECOND as u128);
253         *prev_amount = prev_amount.saturating_sub(amount);
254         Some(MultiAsset {
255             fun: Fungibility::Fungible(amount),
256             id: XcmAssetId::Concrete(id.clone()),
257         })
258     } else {
259         None
260     }
261 }
```

- Function `buy_weight`:

Listing 14: primitives/manta/src/xcm.rs (Line 183)

```
164 match (first_asset.id, first_asset.fun) {
165         (XcmAssetId::Concrete(id), Fungibility::Fungible(_)) => {
166             let asset_id = M::asset_id(&id.clone().into()).ok_or({
167                 log::debug!(
168                     target: "FirstAssetTrader::buy_weight",
169                     "asset_id missing for asset location with id:
  ↳ {:?}",
170                     id,
171                 );
172                 XcmError::TooExpensive
173             })?;
174             let units_per_second = M::units_per_second(&asset_id).
```

DRAFT

DRAFT

```
ok_or({
                log::debug!(
                    target: "FirstAssetTrader::buy_weight",
                    "units_per_second missing for asset with id: {:?}",
                    id,
                );
                XcmError::TooExpensive
            })?;

            let amount = units_per_second * (weight as u128) / (WEIGHT_PER_SECOND as u128);
            // we don't need to proceed if amount is zero.
            // This is very useful in tests.
            if amount.is_zero() {
                return Ok(payment);
            }
            let required = MultiAsset {
                fun: Fungibility::Fungible(amount),
                id: XcmAssetId::Concrete(id.clone()),
            };

```

The following snippets show how the q divisor is calculated and how it's equal to zero if WEIGHT_PER_SECOND is zero too.

- q is equal to 100 * Balance::from(ExtrinsicBaseWeight::get());
- ExtrinsicBaseWeight is equal to 86_298 * WEIGHT_PER_NANOS.
- The value of WEIGHT_PER_NANOS is the result of dividing WEIGHT_PER_SECOND several times. If WEIGHT_PER_SECOND is equal to zero, this value will be zero too.

Listing 15: runtime/manta/src/fee.rs (Lines 47,52)

```
pub struct WeightToFee;
impl WeightToFeePolynomial for WeightToFee {
    type Balance = Balance;
    fn polynomial() -> WeightToFeeCoefficients<Self::Balance> {
        // in Polkadot, extrinsic base weight (smallest non-zero weight) is mapped to 1/10 CENT:
        // in Manta Parachain, we map to 1/10 of that, or 1/100
```

```
  ↳ CENT
45         // TODO, revisit here to figure out why use this
  ↳ polynomial
46         let p = currency::cMANTA;
47         let q = 100 * Balance::from(ExtrinsicBaseWeight::get());
48         smallvec![WeightToFeeCoefficient {
49             degree: 1,
50             negative: false,
51             coeff_frac: Perbill::from_rational(p % q, q),
52             coeff_integer: p / q,
53         }]
54     }
55 }
56
```

Listing 16: runtime/common/src/lib.rs (Line 115)

```
101 parameter_types! {
102     /// Time to execute a NO-OP extrinsic, for example `System::
  ↳ remark`.
103     /// Calculated by multiplying the *Average* with `1` and
  ↳ adding `0`.
104     ///
105     /// Stats nanoseconds:
106     ///   Min, Max: 86_060, 86_999
107     ///   Average:  86_298
108     ///   Median:   86_248
109     ///   Std-Dev:  207.19
110     ///
111     /// Percentiles nanoseconds:
112     ///   99th: 86_924
113     ///   95th: 86_828
114     ///   75th: 86_347
115     pub const ExtrinsicBaseWeight: Weight = 86_298 *
  ↳ WEIGHT_PER_NANOS;
116 }
```

Listing 17: primitives/manta/src/constants.rs (Line 110)

```
109 /// 1_000_000_000_000
110 pub const WEIGHT_PER_SECOND: Weight = 1_000_000_000_000;
111 /// 1_000_000_000
112 pub const WEIGHT_PER_MILLIS: Weight = WEIGHT_PER_SECOND / 1000;
```

DRAFT

```
/// 1_000_000
pub const WEIGHT_PER_MICROS: Weight = WEIGHT_PER_MILLIS / 1000;
/// 1_000
pub const WEIGHT_PER_NANOS: Weight = WEIGHT_PER_MICROS / 1000;
```

Recommendation:

We recommend a review of these identified areas to ensure that adequate arithmetic checks are in place and a safety constraint is set for WEIGHT_PER_SECOND to prevent it from reaching zero. These improvements will further fortify the system, ensuring stability, reliability, and secure operation.

- It is recommended to add a constraint to ensure that WEIGHT_PER_SECOND is never 0.
- In “release” mode, Rust does not panic! due to overflows and overflowed values simply “wrap” without any explicit feedback to the user. It is recommended to use vetted safe math libraries for arithmetic operations consistently throughout the smart contract system. Consider replacing the multiplication operator with Rust’s checked_mul method, the subtraction operator with Rust’s checked_subs method, and so on.

DRAFT

FUZZING

DRAFT

5.1 FUZZ TESTS

We aimed to rigorously evaluate the performance of the manta-sbt pallet when subjected to unforeseen and arbitrarily produced input data. Our objective was to identify any potential anomalies that could lead to system crashes and subsequently a Denial of Service (DoS). For the execution of these tests, we employed the honggfuzz tool to propagate the randomly generated inputs towards the pallet's public functions. Throughout the testing process, the code remained stable without any instances of panic. Our testing methodology was centered around three primary scenarios, with the ultimate goal of evaluating and strengthening the robustness of the manta-sbt pallet in diverse conditions.

The three main scenarios that were checked were the following:

- We tried to crash the pallet using random data as the zkSBT metadata in the to_private function

Listing 18: pallets/manta-sbt/manta-sbt-fuzzing/src/main.rs (Line 188)

```
188 loop {
189         /*-----------------------------------------------------*/
190         Balances::set_balance(Origin::root(), ALICE, 1_000_000_000_000_000, 0);
191         fuzz!(|data: [u8; 201] | {
192 
193             let vec = data.to_vec();
194             let bounded_vec: Result<BoundedVec<u8, ConstU32<200>>, ()> = vec.try_into();
195             let bounded_vec = bounded_vec.expect("");
196             MantaSBTPallet::reserve_sbt(Origin::signed(ALICE));
197             println!("{:#?}", data);
198             let mut rng = OsRng;
199             let value = 1;
200             let id = field_from_id(1);
201 
202             let post = sample_to_private(id, value, &mut rng);
203             MantaSBTPallet::to_private(Origin::signed(ALICE), Box::new(post), bounded_vec);
204         });
```

DRAFT

```
205     }
```

- We tried to crash the pallet using random data as the zkSBT metadata in the mint_sbt_eth function

Listing 19: pallets/manta-sbt/manta-sbt-fuzzing/src/main.rs (Line 208)

```
208 loop{
209         Balances::set_balance(Origin::root(), ALICE, 1
 ↳ _000_000_000_000_000, 0);
210         fuzz!(|data: [u8; 201]| {
211             let vec = data.to_vec();
212             let bounded_vec: Result<BoundedVec<u8, ConstU32<200>>,
 ↳  ()> = vec.try_into();
213             let bounded_vec = bounded_vec.expect("");
214             let mut rng = OsRng;
215             let value = 1;
216             let id = field_from_id(1);
217             let post = Box::new(sample_to_private(id, value, &mut
 ↳ rng));
218 
219             MantaSBTPallet::change_allowlist_account(Origin::root
 ↳ (), Some(ALICE));
220             let evm_mint_type = EvmAddressType::Bab(MantaSBTPallet
 ↳ ::eth_address(&alice_eth()));
221             Timestamp::set_timestamp(10);
222             MantaSBTPallet::set_mint_chain_info(Origin::root(),
 ↳ MintType::Bab, 0, None);
223             MantaSBTPallet::allowlist_evm_account(Origin::signed(
 ↳ ALICE), evm_mint_type);
224             MantaSBTPallet::mint_sbt_eth(
225                 Origin::signed(ALICE),
226                 post.clone(),
227                 1,
228                 MantaSBTPallet::eth_sign(&alice_eth(), &post.proof
 ↳ , 1),
229                 evm_mint_type,
230                 Some(0),
231                 Some(0),
232                 Some(bounded_vec),
233             );
234         });
```

DRAFT

```
235     }
```

- We tried to crash the pallet using random data as a EVM signature in the mint_sbt_eth function

Listing 20: pallets/manta-sbt/manta-sbt-fuzzing/src/main.rs (Line 237)

```
237 loop {
238         Balances::set_balance(Origin::root(), ALICE, 1
  ↳ _000_000_000_000_000, 0);
239         fuzz!(|data: [u8; 65]| {
240             let vec = data.to_vec();
241             let bounded_vec: Result<BoundedVec<u8, ConstU32<200>>,
  ↳  ()> = vec.try_into();
242             let bounded_vec = bounded_vec.expect("");
243             let mut rng = OsRng;
244             let value = 1;
245             let id = field_from_id(1);
246             let post = Box::new(sample_to_private(id, value, &mut
  ↳ rng));
247 
248             MantaSBTPallet::change_allowlist_account(Origin::root
  ↳ (), Some(ALICE));
249             let evm_mint_type = EvmAddressType::Bab(MantaSBTPallet
  ↳ ::eth_address(&alice_eth()));
250             Timestamp::set_timestamp(10);
251             MantaSBTPallet::set_mint_chain_info(Origin::root(),
  ↳ MintType::Bab, 0, None);
252             MantaSBTPallet::allowlist_evm_account(Origin::signed(
  ↳ ALICE), evm_mint_type);
253             MantaSBTPallet::mint_sbt_eth(
254                 Origin::signed(ALICE),
255                 post.clone(),
256                 1,
257                 data,
258                 evm_mint_type,
259                 Some(0),
260                 Some(0),
261                 Some(bvec![0])
262             );
263         });
264     }
```

DRAFT

MANUAL TESTING

DRAFT

6.1 zkSBTs CANNOT BE MINTED TWICE

We checked if it was possible to mint two zkSBT tokens with the same id. Since this id is not submitted by the user and the only way to get an id is through a call to next_sbt_and_increment function, it's not possible to reserve the same id twice - PASSED

6.2 WEIGHTS ARE ESTIMATED CORRECTLY

We checked that every public function has its weight estimated and there is not any dynamic structure data used as input. Extensive testing was conducted to cover all scenarios, edge cases, and potential attack vectors. The results demonstrated the robustness of the weight calculations under a variety of conditions - PASSED

6.3 zkSBTs CANNOT BE TRANSFERRED

We attempted to use a minted zkSBT as a standard asset, testing if we could integrate a TransferPost from the to_private extrinsic into other operations within the Manta Pay pallet. Both our tests and those conducted by the Manta Network team were unsuccessful; it was not possible to transfer a zkSBT - PASSED

	Sources	Sender Posts	Receiver Posts	Sinks	Sink Accounts
manta-sbt::to_private	1	0	1	0	
manta-pay::to_private	1	0	1	0	0
manta-pay::to_public	0	2	1	1	1
manta-pay::private_transfer	0	2	2	0	0

DRAFT

6.4 ACCESS CONTROL IS IN PLACE

allowlist_evm_account:

We checked if an account that is not the AllowlistAccount can call to allowlist_evm_account and add a new address - PASSED

change_allowlist_account:

We attempted to alter the AllowlistAccount by invoking the change_-allowlist_account function with a regular user - PASSED

set_mint_info:

We tried to change the minting period with a regular user. - PASSED

```
test tests::testing__non_admins_calls_to_set_mint_chain_info_will_fail ... ok
test tests::testing__regular_user_calls_to_allowlist_evm_account_will_fail ... ok
test tests::testing__non_admins_calls_to_change_allowlist_account_will_fail ... ok
test tests::testing__trying_to_privatize_the_zksbt_token_and_transfering_it_fails has been running for over 60 seconds
test tests::testing__trying_to_public_the_minted_token_fails has been running for over 60 seconds
test tests::testing__trying_to_public_the_minted_token_fails ... ok
test tests::testing__trying_to_privatize_the_zksbt_token_and_transfering_it_fails ... ok
```

DRAFT

AUTOMATED TESTING

DRAFT

7.1 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.

ID	package	Short Description
RUSTSEC-2022-0040	owning_ref	Multiple soundness issues in ‘owning_ref‘

Listing 21

```
owning_ref 0.4.1
 prometheus-client 0.16.0
    libp2p-metrics 0.7.0
       libp2p 0.46.1
           sc-telemetry 4.0.0-dev
              sc-sysinfo 6.0.0-dev
                 sc-service 0.10.0-dev
                    try-runtime-cli 0.10.0-dev
                       polkadot-cli 0.9.28
                          manta 4.0.6
                          cumulus-relay-chain-inprocess-interface 0.1.0
                              manta 4.0.6
```

DRAFT

ID	package	Short Description
RUSTSEC-2022-0046	rocksdb	Out-of-bounds read when opening multiple column families with TTL

Listing 22

```
rocksdb 0.18.0
 kvdb-rocksdb 0.15.2
     sc-client-db 0.10.0-dev
        sc-service 0.10.0-dev
           try-runtime-cli 0.10.0-dev
              polkadot-cli 0.9.28
                 manta 4.0.6
                 cumulus-relay-chain-inprocess-interface 0.1.0
                     manta 4.0.6
              manta 4.0.6
           sc-cli 0.10.0-dev
```

DRAFT

ID	package	Short Description
RUSTSEC-2020-0071	time	Potential segfault in the time crate

Listing 23

```
time 0.1.45
 chrono 0.4.24
     tracing-subscriber 0.2.25
        sp-tracing 5.0.0
           sp-runtime-interface 6.0.0
              sp-tasks 4.0.0-dev
                 sc-executor 0.10.0-dev
                     try-runtime-cli 0.10.0-dev
                        polkadot-cli 0.9.28
                           manta 4.0.6
                           cumulus-relay-chain-inprocess-interface 0.1.0
                               manta 4.0.6
                        manta 4.0.6
```

ID	package	Short Description
RUSTSEC-2022-0075	wasmtime	Bug in pooling instance allocator
RUSTSEC-2022-0076	wasmtime	Bug in Wasmtime implementation of pooling instance allocator

Listing 24

```
wasmtime 0.38.3
 sp-wasm-interface 6.0.0
    sp-sandbox 0.10.0-dev
       sc-executor-wasmtime 0.10.0-dev
          sc-executor 0.10.0-dev
             try-runtime-cli 0.10.0-dev
                polkadot-cli 0.9.28
                   manta 4.0.6
                   cumulus-relay-chain-inprocess-interface 0.1.0
                       manta 4.0.6
```

DRAFT

7.2 UNSAFE RUST CODE DETECTION

Description:

Halborn used automated security scanners to assist with the detection of well-known security issues and vulnerabilities. Among the tools used was cargo-geiger, a security tool that lists statistics related to the usage of unsafe Rust code in a core Rust codebase and all its dependencies.

```
Symbols:
    🔒  = No `unsafe` usage found, declares #![forbid(unsafe_code)]
    ?   = No `unsafe` usage found, missing #![forbid(unsafe_code)]
    ☢   = `unsafe` usage found

Functions  Expressions  Impls  Traits  Methods  Dependency

0/0        0/0          0/0    0/0     0/0      ?  pallet-manta-sbt 4.0.6
15/18      453/460      3/3    0/0     12/12    ☢  ├── anyhow 1.0.71
7/22       216/792      2/6    0/0     2/5      ☢  │   └── backtrace 0.3.67
0/0        5/20         0/0    0/0     0/0      ☢  │       ├── addr2line 0.19.0
0/0        0/0          0/0    0/0     0/0      ?  │       │   ├── fallible-iterator 0.2.0
1/1        38/57        1/3    1/1     0/0      ☢  │       │   ├── gimli 0.27.2
0/0        0/0          0/0    0/0     0/0      ?  │       │   │   ├── fallible-iterator 0.2.0
0/0        41/46        1/1    0/0     0/0      ☢  │       │   │   ├── indexmap 1.9.3
1/1        1241/1367    21/24  1/1     62/69    ☢  │       │   │   │   ├── hashbrown 0.12.3
```

```
Symbols:
    🔒  = No `unsafe` usage found, declares #![forbid(unsafe_code)]
    ?   = No `unsafe` usage found, missing #![forbid(unsafe_code)]
    ☢   = `unsafe` usage found

Functions  Expressions  Impls  Traits  Methods  Dependency

0/0        0/0          0/0    0/0     0/0      ?  pallet-manta-support 4.0.6
15/18      453/460      3/3    0/0     12/12    ☢  ├── anyhow 1.0.71
7/22       216/792      2/6    0/0     2/5      ☢  │   └── backtrace 0.3.67
0/0        5/20         0/0    0/0     0/0      ☢  │       ├── addr2line 0.19.0
0/0        0/0          0/0    0/0     0/0      ?  │       │   ├── fallible-iterator 0.2.0
1/1        38/57        1/3    1/1     0/0      ☢  │       │   ├── gimli 0.27.2
0/0        0/0          0/0    0/0     0/0      ?  │       │   │   ├── fallible-iterator 0.2.0
0/0        41/46        1/1    0/0     0/0      ☢  │       │   │   ├── indexmap 1.9.3
```

```
Symbols:
    🔒  = No `unsafe` usage found, declares #![forbid(unsafe_code)]
    ?   = No `unsafe` usage found, missing #![forbid(unsafe_code)]
    ☢   = `unsafe` usage found

Functions  Expressions  Impls  Traits  Methods  Dependency

0/0        0/0          0/0    0/0     0/0      ?  manta 4.0.6
0/4        0/6          0/1    0/3     0/0      ?  ├── async-trait 0.1.68
0/0        15/15        0/0    0/0     3/3      ☢  │   ├── proc-macro2 1.0.58
0/0        4/4          0/0    0/0     0/0      ☢  │   │   └── unicode-ident 1.0.8
0/0        0/0          0/0    0/0     0/0      ?  │   ├── quote 1.0.27
0/0        15/15        0/0    0/0     3/3      ☢  │   │   └── proc-macro2 1.0.58
0/0        79/79        3/3    0/0     2/2      ☢  │   └── syn 2.0.16
0/0        15/15        0/0    0/0     3/3      ☢  │       ├── proc-macro2 1.0.58
0/0        0/0          0/0    0/0     0/0      ?  │       ├── quote 1.0.27
0/0        4/4          0/0    0/0     0/0      ☢  │       └── unicode-ident 1.0.8
0/0        0/0          0/0    0/0     0/0      ?  ├── calamari-runtime 4.0.6
0/0        0/0          0/0    0/0     0/0      ?  │   ├── calamari-vesting 4.0.6
0/0        7/7          0/0    0/0     0/0      ☢  │   │   ├── parity-scale-codec 3.5.0
2/2        350/350      2/2    0/0     7/7      ☢  │   │   │   ├── arrayvec 0.7.2
0/0        5/5          0/0    0/0     0/0      ☢  │   │   │   │   └── serde 1.0.163
```

DRAFT

```
Symbols:
    🔒  = No `unsafe` usage found, declares #![forbid(unsafe_code)]
    ❓  = No `unsafe` usage found, missing #![forbid(unsafe_code)]
    ☢️  = `unsafe` usage found

Functions  Expressions  Impls  Traits  Methods  Dependency

0/0        0/0          0/0    0/0     0/0      ❓ manta-primitives 4.0.6
1/1        16/18        1/1    0/0     0/0      ☢️ ├── log 0.4.17
0/0        0/0          0/0    0/0     0/0      ❓ │   ├── cfg-if 1.0.0
0/0        5/5          0/0    0/0     0/0      ☢️ │   ├── serde 1.0.163
0/0        0/0          0/0    0/0     0/0      ❓ │   │   └── serde_derive 1.0.163
0/0        15/15        0/0    0/0     3/3      ☢️ │   │       ├── proc-macro2 1.0.58
0/0        4/4          0/0    0/0     0/0      ☢️ │   │       │   └── unicode-ident 1.0.8
0/0        0/0          0/0    0/0     0/0      ❓ │   │       ├── quote 1.0.27
0/0        15/15        0/0    0/0     3/3      ☢️ │   │       │   └── proc-macro2 1.0.58
0/0        79/79        3/3    0/0     2/2      ☢️ │   │       └── syn 2.0.16
0/0        15/15        0/0    0/0     3/3      ☢️ │   │           ├── proc-macro2 1.0.58
0/0        0/0          0/0    0/0     0/0      ❓ │   │           ├── quote 1.0.27
```

```
Symbols:
    🔒  = No `unsafe` usage found, declares #![forbid(unsafe_code)]
    ❓  = No `unsafe` usage found, missing #![forbid(unsafe_code)]
    ☢️  = `unsafe` usage found

Functions  Expressions  Impls  Traits  Methods  Dependency

0/0        0/0          0/0    0/0     0/0      ❓ session-key-primitives 4.0.6
0/0        0/0          0/0    0/0     0/0      ❓ ├── manta-primitives 4.0.6
1/1        16/18        1/1    0/0     0/0      ☢️ │   ├── log 0.4.17
0/0        0/0          0/0    0/0     0/0      ❓ │   │   ├── cfg-if 1.0.0
0/0        5/5          0/0    0/0     0/0      ☢️ │   │   ├── serde 1.0.163
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   └── serde_derive 1.0.163
0/0        15/15        0/0    0/0     3/3      ☢️ │   │   │       ├── proc-macro2 1.0.58
0/0        4/4          0/0    0/0     0/0      ☢️ │   │   │       │   └── unicode-ident 1.0.8
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │       ├── quote 1.0.27
```

```
Symbols:
    🔒  = No `unsafe` usage found, declares #![forbid(unsafe_code)]
    ❓  = No `unsafe` usage found, missing #![forbid(unsafe_code)]
    ☢️  = `unsafe` usage found

Functions  Expressions  Impls  Traits  Methods  Dependency

0/0        0/0          0/0    0/0     0/0      ❓ calamari-runtime 4.0.6
0/0        0/0          0/0    0/0     0/0      ❓ ├── calamari-vesting 4.0.6
0/0        7/7          0/0    0/0     0/0      ☢️ │   ├── parity-scale-codec 3.5.0
2/2        350/350      2/2    0/0     7/7      ☢️ │   │   ├── arrayvec 0.7.2
0/0        5/5          0/0    0/0     0/0      ☢️ │   │   │   └── serde 1.0.163
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │       └── serde_derive 1.0.163
0/0        15/15        0/0    0/0     3/3      ☢️ │   │   │           ├── proc-macro2 1.0.58
0/0        4/4          0/0    0/0     0/0      ☢️ │   │   │           │   └── unicode-ident 1.0.8
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │           ├── quote 1.0.27
```

```
Symbols:
    🔒  = No `unsafe` usage found, declares #![forbid(unsafe_code)]
    ❓  = No `unsafe` usage found, missing #![forbid(unsafe_code)]
    ☢️  = `unsafe` usage found

Functions  Expressions  Impls  Traits  Methods  Dependency

0/0        0/0          0/0    0/0     0/0      ❓ runtime-common 4.0.6
0/0        0/0          0/0    0/0     0/0      ❓ ├── manta-primitives 4.0.6
1/1        16/18        1/1    0/0     0/0      ☢️ │   ├── log 0.4.17
0/0        0/0          0/0    0/0     0/0      ❓ │   │   ├── cfg-if 1.0.0
0/0        5/5          0/0    0/0     0/0      ☢️ │   │   ├── serde 1.0.163
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   └── serde_derive 1.0.163
0/0        15/15        0/0    0/0     3/3      ☢️ │   │   │       ├── proc-macro2 1.0.58
0/0        4/4          0/0    0/0     0/0      ☢️ │   │   │       │   └── unicode-ident 1.0.8
```

DRAFT

```
Symbols:
    🔒  = No `unsafe` usage found, declares #![forbid(unsafe_code)]
    ❓  = No `unsafe` usage found, missing #![forbid(unsafe_code)]
    ☢️  = `unsafe` usage found

Functions  Expressions  Impls  Traits  Methods  Dependency

0/0        0/0          0/0    0/0     0/0      ❓ dolphin-runtime 4.0.6
0/0        0/0          0/0    0/0     0/0      ❓ ├── cumulus-pallet-dmp-queue 0.1.0
0/0        0/0          0/0    0/0     0/0      ❓ │   ├── cumulus-primitives-core 0.1.0
0/0        7/7          0/0    0/0     0/0      ☢️ │   │   ├── parity-scale-codec 3.5.0
2/2        350/350      2/2    0/0     7/7      ☢️ │   │   │   ├── arrayvec 0.7.2
0/0        5/5          0/0    0/0     0/0      ☢️ │   │   │   │   └── serde 1.0.163
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │       └── serde_derive 1.0.163
0/0        15/15        0/0    0/0     3/3      ☢️ │   │   │   │           ├── proc-macro2 1.0.58
0/0        4/4          0/0    0/0     0/0      ☢️ │   │   │   │           │   └── unicode-ident 1.0.8
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │           ├── quote 1.0.27
Symbols:
    🔒  = No `unsafe` usage found, declares #![forbid(unsafe_code)]
    ❓  = No `unsafe` usage found, missing #![forbid(unsafe_code)]
    ☢️  = `unsafe` usage found

Functions  Expressions  Impls  Traits  Methods  Dependency

0/0        0/0          0/0    0/0     0/0      ❓ manta-runtime 4.0.6
0/0        0/0          0/0    0/0     0/0      ❓ ├── cumulus-pallet-dmp-queue 0.1.0
0/0        0/0          0/0    0/0     0/0      ❓ │   ├── cumulus-primitives-core 0.1.0
0/0        7/7          0/0    0/0     0/0      ☢️ │   │   ├── parity-scale-codec 3.5.0
2/2        350/350      2/2    0/0     7/7      ☢️ │   │   │   ├── arrayvec 0.7.2
0/0        5/5          0/0    0/0     0/0      ☢️ │   │   │   │   └── serde 1.0.163
0/0        0/0          0/0    0/0     0/0      ❓ │   │   │   │       └── serde_derive 1.0.163
0/0        15/15        0/0    0/0     3/3      ☢️ │   │   │   │           ├── proc-macro2 1.0.58
0/0        4/4          0/0    0/0     0/0      ☢️ │   │   │   │           │   └── unicode-ident 1.0.8
```

DRAFT

THANK YOU FOR CHOOSING

// HALBORN

DRAFT