Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

MANTA NETWORK

Manta Chain (SBT)

Veridise Inc.

June 18, 2023

► **Prepared For:**

Manta Network
https://manta.network/

► **Prepared By:**

Shankara Pailoor
Ben Sepanski
Burak Kadron
Jon Stephens

► **Contact Us:** contact@veridise.com

► **Version History:**

June. 01, 2023 Initial Draft

Contents

1 Executive Summary

From May. 5, 2023 to May. 26, 2023, Manta Network engaged Veridise to review the security of their NFT private offering protocol called MantaSBT, whereby clients (such as other parachains or end-users) can privately mint different NFT or SBT tokens. Veridise conducted the assessment over 12 person-weeks, with 4 engineers reviewing code over 3 weeks on commit `ceb9e46`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers, namely static analysis and fuzz testing, as well as extensive manual auditing.

Code assessment. The MantaSBT protocol is implemented as a pallet in the Manta substrate blockchain, henceforth referred to as Manta Chain. Since Manta Chain is developed fully open-source, Manta Network developers provided Veridise auditors the link to Manta Chain's public github repository along with the commit to be reviewed and the pallets relevant to the MantaSBT protocol. In addition to the code, Veridise auditors were given some documents outlining the intended behavior of the MantaSBT including a document from their website along with a medium article. The code relevant to MantaSBT is reasonably well documented and contains detailed, yet clear, comments specifying the intended behavior of the corresponding code.

Summary of issues detected. The audit uncovered 6 issues, 0 of which are assessed to be of high or critical severity by the Veridise auditors. The Veridise auditors also identified 1 issue which was assessed as medium-severity including V-MSBT-VUL-001 which could result in users losing funds.

Recommendations. While none of the issues discovered in this audit were assessed as critical or high-severity, we still recommend that they address the medium severity issue found in the audit.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Manta Chain (SBT)	`ceb9e46`	Rust	Substrate

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
May. 5 - May. 26, 2023	Manual & Tools	4	12 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	0	0
High-Severity Issues	0	0
Medium-Severity Issues	1	1
Low-Severity Issues	1	1
Warning-Severity Issues	3	3
Informational-Severity Issues	1	1
TOTAL	6	6

Table 2.4: Category Breakdown.

Name	Number
Logic Error	2
Data Validation	2
Bad Extrinsic Weight	1
Gas Optimization	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Manta Network's protocol MantaSBT, which allows users to privately mint NFTs or SBTs. The corresponding private assets are referred to as zkSBTs. This protocol is very similar to MantaPay, another protocol of Manta Network; however, unlike MantaPay, MantaSBT prohibits users from trading the privately minted assets. As such, in our audit, we sought to answer the following questions:

- Are there any attack vectors whereby attackers can learn information which should be kept private by the protocol i.e, which user mints a particular NFT?
- Is there a way for users to transfer minted zkSBTs?
- Can users mint two identical zkSBTs?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* We used `cargo-audit`, an open-source static analysis tool used to audit Cargo.lock files for crates with security vulnerabilities reported to the RustSec Advisory Database.
- *Fuzzing/Property-based Testing.* We leveraged fuzz testing to determine if MantaSBT's implementation deviated from the intended behavior. To do this, we first encoded *invariants*, logical formulas that should hold throughout MantaSBT's lifecycle, as assertions. We then wrote harnesses for afl.rs, which generated random sequences of external calls relevant to those assertions. If afl.rs found a crash then this indicated either a panic or a violation of the invariant. We provide a table outlining the invariants we fuzzed tested as well as the fuzzing methodology in Chapter 5.

Scope. The scope of the audit was limited to all the code in the following top-level directories of Manta Chain:

- `node/*` - starts up a node in Manta Chain.
- `pallets/manta-sbt` - contains the MantaSBT logic.
- `pallets/manta-support` - includes various helper functions.
- `primitives/*` - defines traits and types used by other packages.
- `runtime/*` - contains runtime configuration for different execution environments.

Veridise auditors first reviewed previous audit reports of substrate blockchains, the documentation provided by Manta Network developers, and inspected the provided tests to determine what logic had been extensively tested. They then began a manual audit of the code assisted by

both static analyzers and property-based fuzz testing. During the audit, the Veridise auditors met with the Manta Network developers on a weekly basis and messaged over Telegram to ask questions about the code and report suspected bugs.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowleged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-MSBT-VUL-001	SBT reservations can be overwritten	Medium	Fixed
V-MSBT-VUL-002	Extrinsics charge static fees that do not account for...	Low	Fixed
V-MSBT-VUL-003	Missing validation in pull_ledger_diff	Warning	Acknowledged
V-MSBT-VUL-004	Off-by-one error in to_private	Warning	Acknowledged
V-MSBT-VUL-005	Unnecessary Storage Variable	Warning	Fixed
V-MSBT-VUL-006	Missing validation when setting mint info	Info	Acknowledged

4.1 Detailed Description of Issues

4.1.1 V-MSBT-VUL-001: SBT reservations can be overwritten

Severity	Medium	Commit	ceb9e46
Type	Logic Error	Status	Fixed
File(s)	pallets/manta-sbt/src/lib.rs		
Location(s)	reserve_sbt		

The manta-sbt pallet exposes a method `reserve_sbt` whereby callers pay MANTA tokens to reserve the right to mint `N` SBTs. This reservation is enforced by allocating reservation ids to the caller of the method so that when users mint an SBT, Manta Chain uses one of the reserved ids to track the number reserved. In particular, Manta Chain maintains a map called `ReservedIds` which maps users to an interval of reservation ids such that the length of the interval indicates the number of SBTs they can mint.

However, `reserve_sbt` does not check whether a user has already reserved SBTs to mint when calling the method and simply updates the mapping to a new interval of length `N`. This can be seen in the method implementation below:

```
/// Reserves AssetIds to be used subsequently in ‘to_private‘ above.
///
/// Increments AssetManager’s AssetId counter.
#[pallet::call_index(1)]
#[pallet::weight(<T as pallet::Config>::WeightInfo::reserve_sbt())]
#[transactional]
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>, _>>()?;

  // The range of ‘AssetIds‘ that are reserved as SBTs
  let start_id: StandardAssetId = *asset_id_range.first().ok_or(Error::<T>::ZeroMints
    )?;
  let stop_id: StandardAssetId = *asset_id_range.last().ok_or(Error::<T>::ZeroMints)
    ?;

  ReservedIds::<T>::insert(&who, (start_id, stop_id));
  Self::deposit_event(Event::<T>::SBTReserved {
      who,
```

```
30        start_id,
31        stop_id,
32    });
33    Ok(())
34  }
```

Thus, if a user calls the method `M` times to reserve `M*N` SBTs, then they will only be able to mint `N`.

Impact Users can lose money because they may think they have reserved `M*N` SBTs when they in-fact can only mint `N`. Furthermore, if a user calls `reserve_sbt` `M` times in a row, then `M*(N-1)` SBTs can no longer be minted. This is due to the fact that reserved ids are always incremented.

For example, suppose someone sets up a relayer account which users could use to purchase zkSBTs for themselves without being linked to the transaction. That relayer account would only receive SBTs on its last call `reserve_sbt`. Implementations in which the relayer reserves and sends the SBTs separately may be error-prone.

Recommendation If at most `N` SBTs can be reserved at a time for a user, then `reserve_sbt` should check whether a user has already reserved SBTs. If the protocol permits more than `N` SBTs to be reserved at a time for a user, then the mapping should be changed. One option might be to map each user to a set of intervals corresponding to ids reserved for them.

Developer Response The developers have acknowledged the issue and a fix has been proposed in this pull request. The fix changes the function to revert if the user has SBT ids already reserved.

4.1.2 V-MSBT-VUL-002: Extrinsics charge static fees that do not account for Merkle tree updates

Severity	Low	**Commit**	ceb9e46
Type	Bad Extrinsic Weight	**Status**	Fixed
File(s)	`pallets/manta-sbt/src/lib.rs`		
Location(s)	to_private, mint_sbt_eth		

Transactions `to_private` and `mint_sbt_eth` take membership proofs and store UTXOs to a Merkle tree on the ledger.

`manta-sbt` shards this Merkle tree into 256 buckets where each bucket has its own Merkle tree. Instead of storing the entire tree at each bucket, the Ledger just stores the last path added to the tree. When adding a UTXO, the Ledger first computes its corresponding bucket, then computes the new path pointing to that UTXO, and finally adds that path to the bucket.

Computing the new path should take time proportional to `log(n)` where `n` is the size of the Merkle Tree. The current benchmarking scheme only covers cases where the previous path is small i.e, at most size 1. However, if the number of transactions gets large i.e, is on the order of hundreds of millions or billions, then the size of the path can get to 24-28 (taking shards into account). If the tree grows to this size, this means each execution of the extrinsic will perform 24-28 hashes, multiplied by the number of UTXOs to be added.

The benchmarking scheme should take into account the size of the tree to make sure that the existing weights are enough to offset the computation of the new Merkle tree path.

Impact In general, it is important to set the weights to account for both computation and storage; setting the weight too low can allow users to perform a large number of transactions with little cost. In particular, malicious users may take advantage of the low fee to launch a DOS attack.

Recommendation We recommend that the weights be computed with a larger database that reflects the state of the chain after a year's worth of use.

Developer Response In progress open PR for manta-pay right now that will be extended to pallet SBT. The manta-pay PR can be found here.

4.1.3 V-MSBT-VUL-003: Missing validation in pull_ledger_diff

Severity	Warning	**Commit**	ceb9e46
Type	Data Validation	**Status**	Acknowledged
File(s)	`pallets/manta-sbt/src/lib.rs`		
Location(s)	pull_ledger_diff		

`pull_ledger_diff` takes as input a `Checkpoint` which is a struct of two fields `receiver_index` and `sender_index` and pulls receiver data from the ledger starting at `receiver_index` up till at most `reciever_index + PULL_MAX_RECEIVER_UPDATE_SIZE`. However, there is no check that this sum cannot overflow for both the sender and receiver index in `pull_receivers` and `pull_receivers_for_shard`.

Impact If the code is compiled without the `--release` flag then a malicious user could crash the node by passing in bad values. If it is built with `--release` then the call will be reported as successful and no senders or receivers will be returned. However, if a benign end user is calling the API with incorrect indexes it might be better to return an error informing them that the index is invalid.

Recommendation We recommend adding bounds checks to be safe and to return an Error.

Developer Response The developers acknowledged the issue and will fix this prior to release.

4.1.4 V-MSBT-VUL-004: Off-by-one error in to_private

Severity	Warning	**Commit**	ceb9e46
Type	Logic Error	**Status**	Acknowledged
File(s)	`pallets/manta-sbt/src/lib.rs`		
Location(s)	to_private		

The `manta-sbt` pallet has a user-callable extrinsic, `to_private`, which allows users to mint reserved SBT tokens. To facilitate minting tokens, the pallet maintains a mapping from users to an interval $[l, u]$ where $l \leq u$ and l and u refer to minimum and maximum asset ids that can be minted by the user. The function does two things. First, it mints an asset with id `l` and then updates the interval to `[l + 1, u]`. If `l + 1 > u`, it removes the user from the map since this indicates they don't have any more user ids to reserve.

There is an edge case where `l = u = 2**128 - 1` where this function's behavior is incorrect. In this case, the extrinsic will revert because `l + 1` would result in an overflow as it is of type `u128`. However, this prevents the user from minting a token reserved for them.

Currently this should not be a problem because it is unlikely that all 2^{128} ids will get reserved anytime soon.

Impact A user may not be able to mint an SBT token reserved for them.

Recommendation We recommend adding an edge case for when `l = 2^128-1` . In that case, the token should be minted and user removed from the map.

Developer Response Not to be resolved. What the auditors point out is correct that the last id will not get minted; however, the resolution is unnecessary as once we hit this edge case the entire zkSBT protocol is unable to function. Furthermore, this value is extremely large and unlikely to ever be reached.

4.1.5 V-MSBT-VUL-005: Unnecessary Storage Variable

Severity	Warning	**Commit**	ceb9e46
Type	Gas Optimization	**Status**	Fixed
File(s)	`pallets/manta-sbt/src/lib.rs`		
Location(s)	382		

The storage variable `UtxoAccumulatorOutputs` keeps track of all Merkle roots generated by `manta-sbt` ; however, this variable is unnecessary. It seems to be taken from the `manta-pay` pallet which uses it to keep track of previous Merkle roots so that when users provide membership proofs for asset transfers, the ledger can check that the root provided was a legitimate one. However, the tokens in `manta-sbt` are non-transferable so there isn't any logic which should use it. In particular, the only two places which use this storage variable are in the functions `has_matching_utxo_accumulator_output` and `register_all`. The former will never be invoked in `manta-sbt` because the asset is non-transferable. The later just adds a newly created Merkle root to `UtxoAccumulatorOutputs`.

Since this storage variable keeps track of every root generated, it is a non-trivial amount of storage to keep on chain. Moreover, every minting transaction, namely `to_private` and `mint_sbt_eth` will incur an extra cost of writing to storage, which is unnecessary.

Impact Unnecessarily high transaction fees for every mint transaction and bloated storage.

Recommendation We recommend removing this storage variable.

Developer Response This has been fixed in the following commit.

4.1.6 V-MSBT-VUL-006: Missing validation when setting mint info

Severity	Info	**Commit**	ceb9e46
Type	Data Validation	**Status**	Acknowledged
File(s)	`pallets/manta-sbt/src/lib.rs`		
Location(s)	new_mint_info, update_mint_info		

When setting metadata for a mint type, the functions `new_mint_info` and `update_mint_info` set the `start_time` and `end_time` associated with the mint type. The functions validate the start and end times by making sure `start_time < end_time`. However, they don't check whether those make sense with respect to the current time. In particular, it seems like you would want `now < end_time`.

Impact Setting `end_time < now` means that nobody can mint for that mint type until it gets changed again.

Recommendation We recommend adding the additional validation which shouldn't be too expensive.

Developer Response This is intended behavior for now. The only way to pause a mint is to set `end_time < now`. This however could be improved by using an `Option<>` and remove the need for this invariant. This is low priority, but we could fix it in the future.

5 Fuzz Testing

5.1 Methodology

Our goal was to fuzz test MantaSBT to assess its functional correctness i.e, whether the implementation deviates from the intended behavior. We used afl.rs as our fuzzer and started writing invariants –logical formulas that should hold after every transaction. We then encoded those invariants as assertions in Rust. For each invariant, we wrote a harness which executed a random sequence of relevant external calls and then asserted the invariant should hold after the calls. We prioritized invariants which had a higher security impact e.g, if violated would allow someone to steal funds. For all invariants, we ran afl.rs for at least 24 hours for each invariant.

5.2 Properties Fuzzed

The following table describes the invariants we fuzz-tested. The first column states which subsystem (e.g, pallet or xcm) the invarianted is associated with. The second describes the invariant informally in English and the last column notes whether we found a bug when fuzzing the invariant (✗ indicates no bug was found and ✓ means fuzzing this invariant revealed a bug). We ran afl.rs for 24 hours when fuzz-testing each invariant.

Table 5.1: Invariants Fuzzed.

Subsystem	Invariant	Bug
manta-sbt	A reserved SBT token should always be mintable	✓
manta-sbt	A minted SBT cannot be transfered via to_private or mint_sbt_eth	✗