Smart contract vulnerables assessment report

Date: 11 December 2024
Project: Open source dynamic assets (Token/ NFT) generator (CIP68)
Version: 1.0
Created by: **Hieu Nguyen Van**

Disclosure

This document contains proprietary information belonging to cardano2vn.io. Duplication, redistribution, or use, in whole or in part, in any form, requires explicit consent from Ancardano2vn.io.

Nonetheless, both the customer cardano2vn and our partners are authorized to share this document with the public to demonstrate security compliance and transparency regarding the outcomes of the Protocol.

Disclaimer and Scope

Our code review represents a snapshot in time, and the findings and recommendations presented in this report reflect the information gathered during the assessment period. It is important to note that any modifications made outside of this timeframe will not be captured in this report.
While diligent efforts have been made to uncover potential vulnerabilities, it is essential to recognize that this assessment may not uncover all potential security issues in the protocol.

It is imperative to understand that the findings and recommendations provided in this audit report should not be construed as investment advice.

Furthermore, it is strongly recommended that projects consider undergoing multiple independent audits and/or participating in bug bounty programs to increase their protocol security.

Please be aware that the scope of this security audit does not extend to the compiler layer, such as the UPLC code generated by the compiler or any areas beyond the audited code.

The scope of the assessment did not include additional creation of unit testing or property-based testing of the contracts.

Version control

Version	Date	Author	Update note
1.0	11 Dec 2024	Tien Nguyen Anh	First draft
1.1	15 Dec 2024	Hieu Nguyen Van	Mint, burn

Assessment overview

This review is intended to confirm that the smart contract features work well and avoid basic bugs before being released to the public for bug hunting.

The review is conducted from December 11, 2024 to December 16, 2024

Assessment components

The assessments are performed only with 02 validators, mint and store, and do not include other related components.

Code base

FUNCTION	MINTING
repository	https://github.com/cardano2vn/cip68generator
commit	09ca522ff1a2778e4fce5ee1e44553f920db19c0
File audit	https://github.com/cardano2vn/cip68generator/blob/main/contract/validators/mint.ak

FUNCTION	UPDATE AND BURNT

repository	https://github.com/cardano2vn/cip68generator
commit	4af7ec9e7e0bf46c2c4bb606c2039b749caf2796
File audit	https://github.com/cardano2vn/cip68generator/blob/main/contract/validators/store.ak

Severity Classification

- **Critical:** This vulnerability has the potential to result in significant financial losses. They often enable attackers to directly steal assets from contracts or users, or permanently lock funds within the contract.
- **Major:** Can lead to damage to the user or platform although the impact may be restricted to specific functionalities or temporal control. Attackers exploiting major vulnerabilities may cause harm or disrupt certain aspects of the platform.
- **Medium**: May not directly result in financial losses, but they can temporarily impair the platform’s functionality. Examples include susceptibility to front-running attacks, which can undermine the integrity of transactions.
- Minor: Minor vulnerabilities do not typically result in financial losses or significant harm to users or the platform. The attack vector may be inconsequential or the attacker’s incentive to exploit it may be minimal.
- **Informational:** These findings do not pose immediate financial risks. These may include platform optimizations, code style recommendations, alignment with naming conventions, overall contract design suggestions, and documentation discrepancies between the code

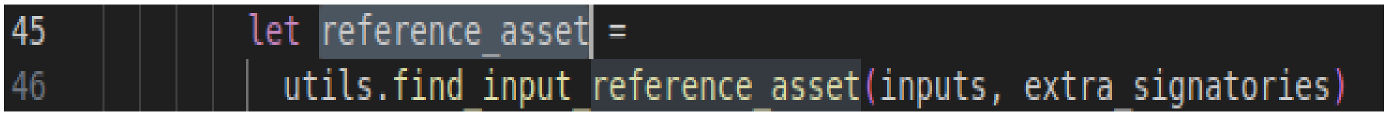
Finding severity ratings

Findings

ID_501-mint.ak

Level	Severity	Location	**Status**
5	Critical	contract/mint.ak	

Description



```
let reference_asset =
  utils.find_input_reference_asset(inputs, extra_signatories)
```

Getting "reference_asset" from "inputs" in "transaction", it is Off-chain reading UTxOs containing NFTs from the "store" smart contract address. Therefore, it does not have the ability to prevent creating NFTs with the same asset-name. At that time, on the "store" there are many NFT refs with the same name, leading to NFT users not being able to correctly to metadata.

Recommendation

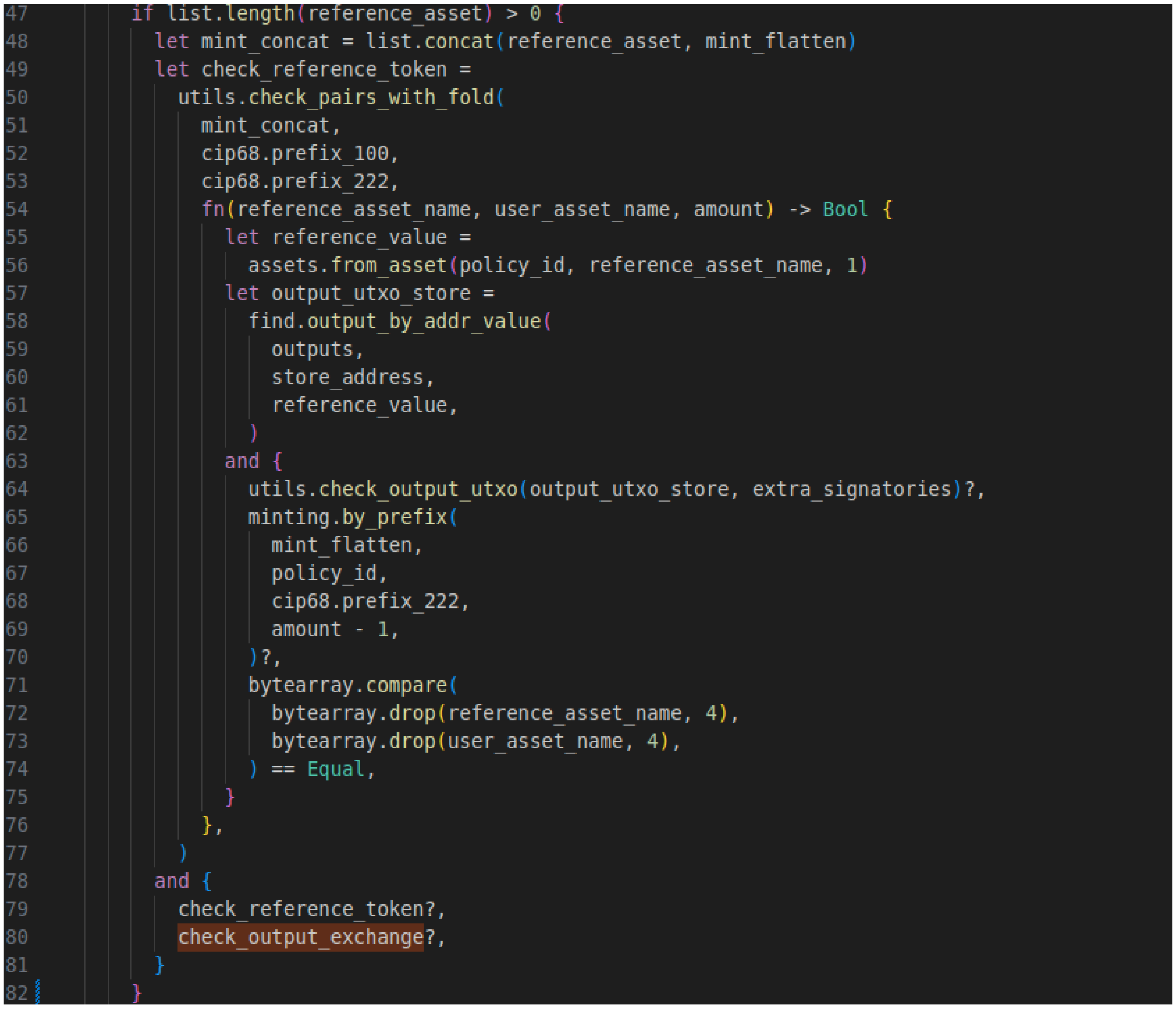
The project team should use Oracle to store previously minted asset-names as a comparison source. Then on the smart contract "mint.ak" will use reference input to read these Asset-names and compare with the minted Asset-names.

ID_301-mint.ak

Level	Severity	Location	**Status**
3	Medium	contract/mint.ak	

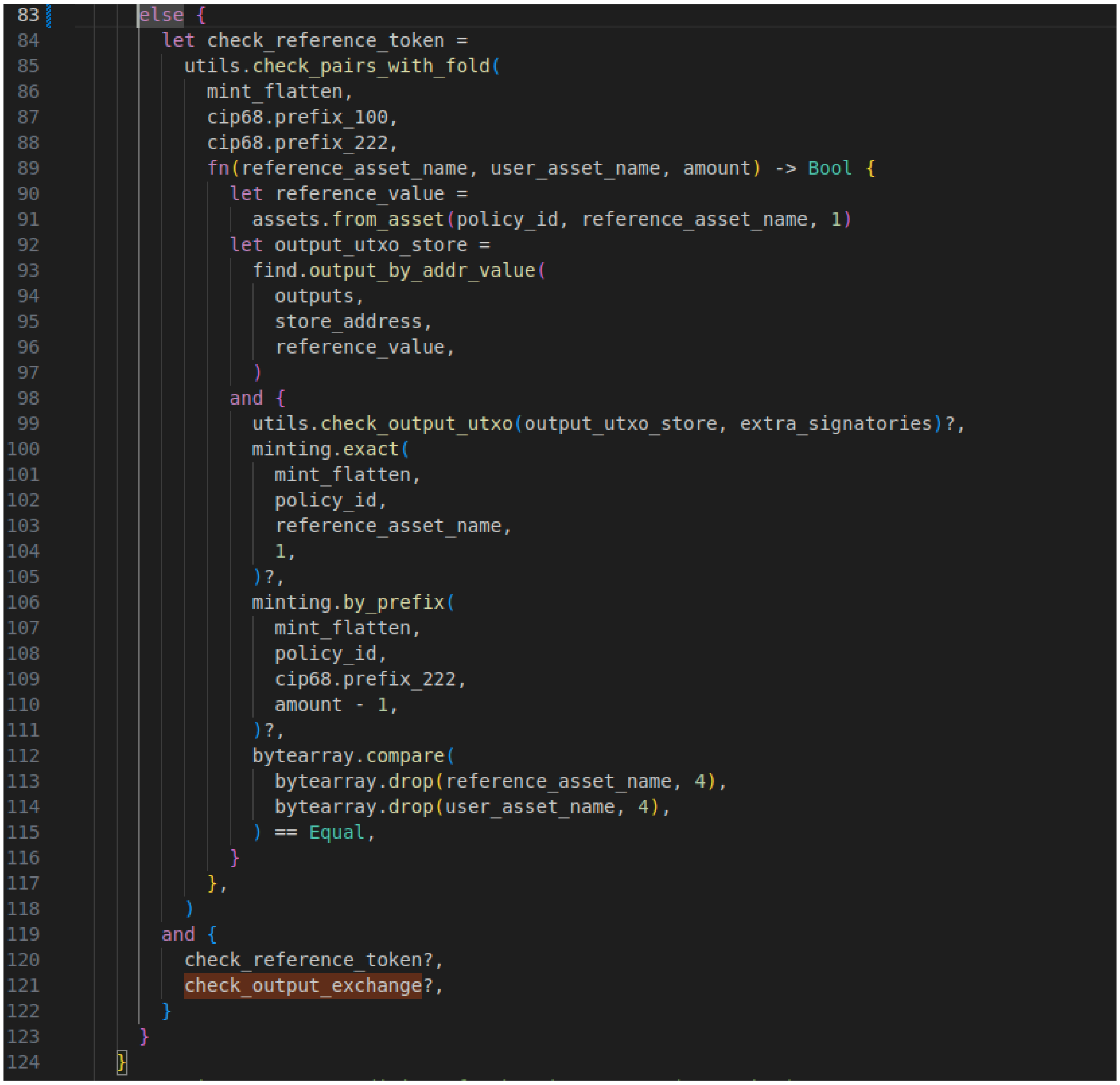
Description

The "if" component



```
    if list.length(reference_asset) > 0 {
      let mint_concat = list.concat(reference_asset, mint_flatten)
      let check_reference_token =
        utils.check_pairs_with_fold(
          mint_concat,
          cip68.prefix_100,
          cip68.prefix_222,
          fn(reference_asset_name, user_asset_name, amount) -> Bool {
            let reference_value =
              assets.from_asset(policy_id, reference_asset_name, 1)
            let output_utxo_store =
              find.output_by_addr_value(
                outputs,
                store_address,
                reference_value,
              )
            and {
              utils.check_output_utxo(output_utxo_store, extra_signatories)?,
              minting.by_prefix(
                mint_flatten,
                policy_id,
                cip68.prefix_222,
                amount - 1,
              )?,
              bytearray.compare(
                bytearray.drop(reference_asset_name, 4),
                bytearray.drop(user_asset_name, 4),
              ) == Equal,
            }
          },
        )
      and {
        check_reference_token?,
        check_output_exchange?,
      }
    }
```

The "else" component


```
        else {
          let check_reference_token =
            utils.check_pairs_with_fold(
              mint_flatten,
              cip68.prefix_100,
              cip68.prefix_222,
              fn(reference_asset_name, user_asset_name, amount) -> Bool {
                let reference_value =
                  assets.from_asset(policy_id, reference_asset_name, 1)
                let output_utxo_store =
                  find.output_by_addr_value(
                    outputs,
                    store_address,
                    reference_value,
                  )
                and {
                  utils.check_output_utxo(output_utxo_store, extra_signatories)?,
                  minting.exact(
                    mint_flatten,
                    policy_id,
                    reference_asset_name,
                    1,
                  )?,
                  minting.by_prefix(
                    mint_flatten,
                    policy_id,
                    cip68.prefix_222,
                    amount - 1,
                  )?,
                  bytearray.compare(
                    bytearray.drop(reference_asset_name, 4),
                    bytearray.drop(user_asset_name, 4),
                  ) == Equal,
                }
              },
            )
          and {
            check_reference_token?,
            check_output_exchange?,
          }
        }
      }
```

These "if" and "else" components are almost the same. This leads to a large smart contract capacity, not optimizing CBOR.

Recommendation

The project team should write the similar parts as variables outside the if else and then and those parts in the if else.

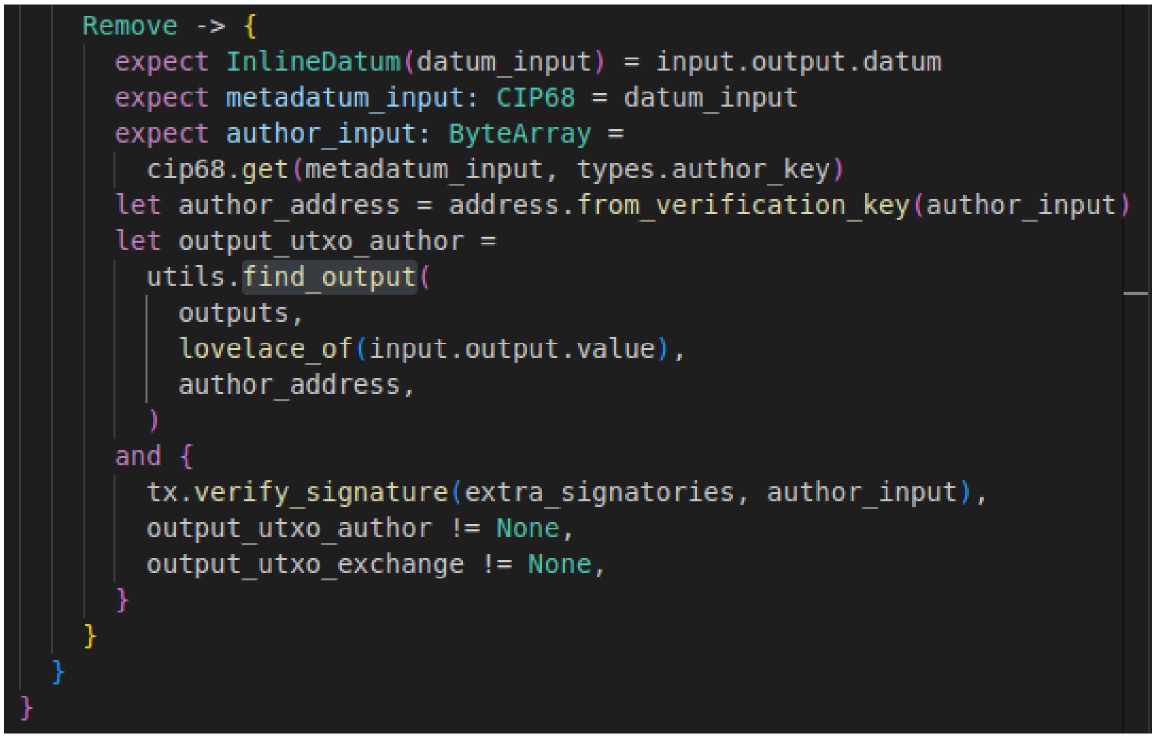
ID_302-mint.ak

Level	Severity	Location	**Status**
3	Medium	contract/store.ak	

Description

- With the Remove condition of the store.ak contract. Here contains the transfer condition to the author's wallet. If the "author" does not burn, can they update the metadata anymore?

- The find_output function only needs 1 utxo of the author to be able to spend other utxos.



```
    Remove -> {
      expect InlineDatum(datum_input) = input.output.datum
      expect metadatum_input: CIP68 = datum_input
      expect author_input: ByteArray =
        cip68.get(metadatum_input, types.author_key)
      let author_address = address.from_verification_key(author_input)
      let output_utxo_author =
        utils.find_output(
          outputs,
          lovelace_of(input.output.value),
          author_address,
        )
      and {
        tx.verify_signature(extra_signatories, author_input),
        output_utxo_author != None,
        output_utxo_exchange != None,
      }
    }
  }
}
```

Recommendation

Add constraints so that users cannot remove or burn other people's assets