ANASTASIA LABS

Security Audit Report

Business Confidential

Date: August 11, 2023
Project: XX5A
Version 1.0

Contents

Scope and Disclaimer

The scope of this audit is limited to the on-chain code specified in the Files audited section below. The code responsible off-chain transaction building was not reviewed in this audit.

Aiken, the language that was used to write the FluidShare contracts is relatively new; as such, the UPLC that it generates may not correctly correspond to the intentions of the Aiken code. The correctness of the Aiken compilation pipeline and the Aiken standard library are both out of the scope of this audit.

Anastasia Labs prioritized identifying the weakest security vectors as well as prominent areas where code quality can be improved. The scope of the audit did not include the creation of additional unit or property-based testing of the contracts.

The findings and recommendations contained herein reflect the information gathered by Anastasia Labs during the course of the assessment, and exclude any changes or modifications made outside of that period.

Assessment overview

From July 18th, 2023 to August 11th, 2023, Fluid Tokens engaged Anastasia Labs to evaluate and conduct a security assessment of their FluidShare Cardano smart contract.

Phases of code auditing activities include the following:

- Planning – Customer goals are gathered.
- Discovery – Perform code review to identify potential vulnerabilities, exploits, possible optimizations and improvements.
- Attack – Confirm potential vulnerabilities through testing.
- Reporting – Document all findings and their resolution status.

The engineering team also conducted a comprehensive review of protocol optimization strategies.

Assessment components

Manual revision

Our manual code auditing is focused on a wide range of attack vectors, including but not limited to.

- UTXO Value Size Spam (Token Dust Attack)
- Large Datum or Unbounded Protocol Datum
- EUTXO Concurrency DoS
- Unauthorized Data modification
- Infinite Mint
- Incorrect Parameterized Scripts
- Other Redeemer
- Other Token Name
- Arbitrary UTXO Datum
- Unbounded protocol value
- Foreign UTXO tokens
- Unspendable UTXOs
- Missing UTXO authentication
- UTXO contention
- Lack of staking control

Code base

Repository

https://github.com/FluidTokens/ft-audit-cardano-sc-v2

Commit

b806d2ffdbc216c14e0695b8f776745729b77d9e

Files audited

SHA256 Checksum	Files
f407e71b5dce7267bdaf65e091de426a5f232392e520b8301461e4961a869bed	ft-sc-renting-v2-master/validators/renting.ak

Finding severity ratings

The following table defines levels of severity and score range that are used throughout the document to assess vulnerability and risk impact.

	Level	Severity	Findings
🟥	5	Critical	0
🟥	4	Major	0
🟧	3	Medium	1
🟨	2	Minor	0
🟦	1	Informational	2

The audit used the Common Vulnerability Scoring System in conjunction with the NVD Calculator to provide a standardised measure for the severity of the identified vulnerabilities. We recognize that many of the metrics considered in CVSS are not relevant for audits of Cardano Smart Contracts; nonetheless, there is still considerable value in adhering to a standard in that it provides more unbiased severity metrics for the findings.

`https://www.first.org/cvss/v3.1/specification-document#Qualitative-Severity-Rating-Scale`

Executive summary

The FluidShare smart contract allows users to rent and borrow NFTs without collateral. More accurately, it allows users to lend and borrow the delegation rights of Cardano native assets. The protocol specifications are:

- While rented, the asset cannot be transferred, listed or sold.
- Once a user has paid to rent a value for some duration, the staking control of the value cannot be revoked or reclaimed until that duration has expired.
- The only person who should be able to reclaim the rented value is the renter (and only while there is no currently active renting term).

ID-301 Double Satisfaction - Unfair Reclaim of Rented Value

	Level	Severity	Status
🟨	3	Medium	Pending

Description

A malicious lender could utilize the double satisfaction vulnerability to reclaim an active loan (that a tenant has paid for) before the deadline. Consider the following case: a lender issues two loans each with the same deadline and the same assets being loaned. A borrower comes along and accepts both of the loans. A malicious lender could build a transaction that unlocks both of the UTxOs containing their lent assets from the script using the `ChangeTermsActive` redeemer and then locks only one of the UTxOs back into the script. This way, they have reclaimed the value they lent to the tenant before the loan's deadline.

Recommendation

We recommend adding the following check to the `ChangeTermsActive` redeemer logic:

```
only_one_sc_input(ctx.transaction.inputs)
```

Resolution

Resolved.

ID-101 Datum as Identifier for Script Inputs

	Level	Severity	Status
🟦	1	Informational	Pending

Description

In various places around the code-base the presence of a Datum was used to identify non-script UTxOs; however, a UTxO can have a Datum irrespective of whether it resides at a script address or a user address.

For instance, consider the following code to identify the user's address:

```
fn get_caller_address(txInputs: List<Input>) {
  let inputList =
    list.filter(
      txInputs,
      fn(input) {
        when input.output.datum is {
          NoDatum -> True
          _ -> False
        }
      },
    )

  expect Some(maybeInput) = list.head(inputList)
  maybeInput.output.address
}
```

If a user's wallet contains UTxOs with Datums in them (for instance, as result of interacting with other protocols such as Encoins or SpaceBudz Marketplace) then this check could prevent them from interacting with the renting contract (unless they have sufficient funds in UTxOs without Datums and the off-chain logic filters to only use UTxOs without datums as inputs for the transaction).

Recommendation

We suggest to use a more accurate means of identifying non-script UTxOs. One such identifier for script UTxOs is to check that the payment credential of the UTxOs address is a ScriptCredential. See:

```
fn get_caller_address(transaction: Transaction) {
  let input_list =
    list.filter(
      transaction.inputs,
      fn(input) {
        when input.output.address.payment_credential is {
          VerificationKeyCredential(_) -> True
          _ -> False
        }
      },
    )

  expect Some(maybe_input) = list.head(input_list)
  maybe_input.output.address
}
```

Resolution

Resolved.

ID-102 Continuing UTxO Value Equality too Strict

	Level	Severity	Status
🟦	1	Informational	Pending

Description

The renting validator checks that the Value in the UTxO being unlocked must be equal to the Value in the continuing output (The UTxO that goes back into the script). This equality check is too strict because minAda can change depending on the size of the UTxO; additionally how minAda is configurable and may change in the future.

```
expect Some(output_to_contract) = list.head(maybe_output)
output_to_contract.value == value_inside
```

This is not a vulnerability because the renter can always use the `WithdrawNFT` redeemer to reclaim their rented value; thus circumventing the `value_is_kept` check.

Recommendation

We suggest to use a less strict equality check when comparing the value in the script input to the value in the continuing output.

```
expect Some(output_to_contract) = list.head(maybe_output)
without_lovelace(output_to_contract.value) == without_lovelace(value_inside)
```

Resolution

Resolved.