Smart Contract Security Assessment

ADAXPRO

January 27, 2022

Version: 1.0

Presented by:

BTBlock LLC

Security level: Public

BT block

Table of Contents

Table of Figures

Table of Tables

Executive summary

Overview

ADAXPRO LTD engaged BtBlock LLC to perform a Security Assessment for its Smart Contracts.

The assessment was conducted remotely by the BTBlock Security Team. Testing took place on December 1, 2021 – January 25, 2022, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks discovered within the environment during the engagement.
- To provide a professional opinion on the security measures' maturity, adequacy, and efficiency.
- To identify potential issues and include improvement recommendations based on the result of our tests.

This report summarises the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the BTBlock Security Teams took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key findings

The following are the major themes and issues identified during the testing period. Within the findings section, these, along with other items, should be prioritized for remediation to reduce the risk they pose.

- KS-ADAXPRO-04 – Exception on an empty list
- KS-ADAXPRO-05 – Partial pattern matching raising an exception
- KS-ADAXPRO-06 – Assumptions made when filtering lists

During the test, the following positive observations were noted regarding the scope of the engagement:

- The team was very supportive and open to discussing the design choices made

We can conclude that the reviewed code implements the documented functionality based on the formal verification and code review.

We can also conclude that we did not find any significant findings with the code during the review.

Scope and rules of engagement

BTBlock performed a Security Assessment for ADAXPRO LTD. The following table documents the targets in scope for the engagement. No other systems or resources were in scope for this assessment.

The source code was supplied through a zip file

Files audited include the following, together with the same project dependencies:

Files included in the code review
. └── src   └── OnChain.hs

Table 1: Scope

Technical analysis & findings

During the Security Assessment for ADAXPRO, we discovered:

- 3 findings with a LOW severity rating.
- 6 findings with an INFORMATIONAL severity rating.

The following chart displays the findings by severity.

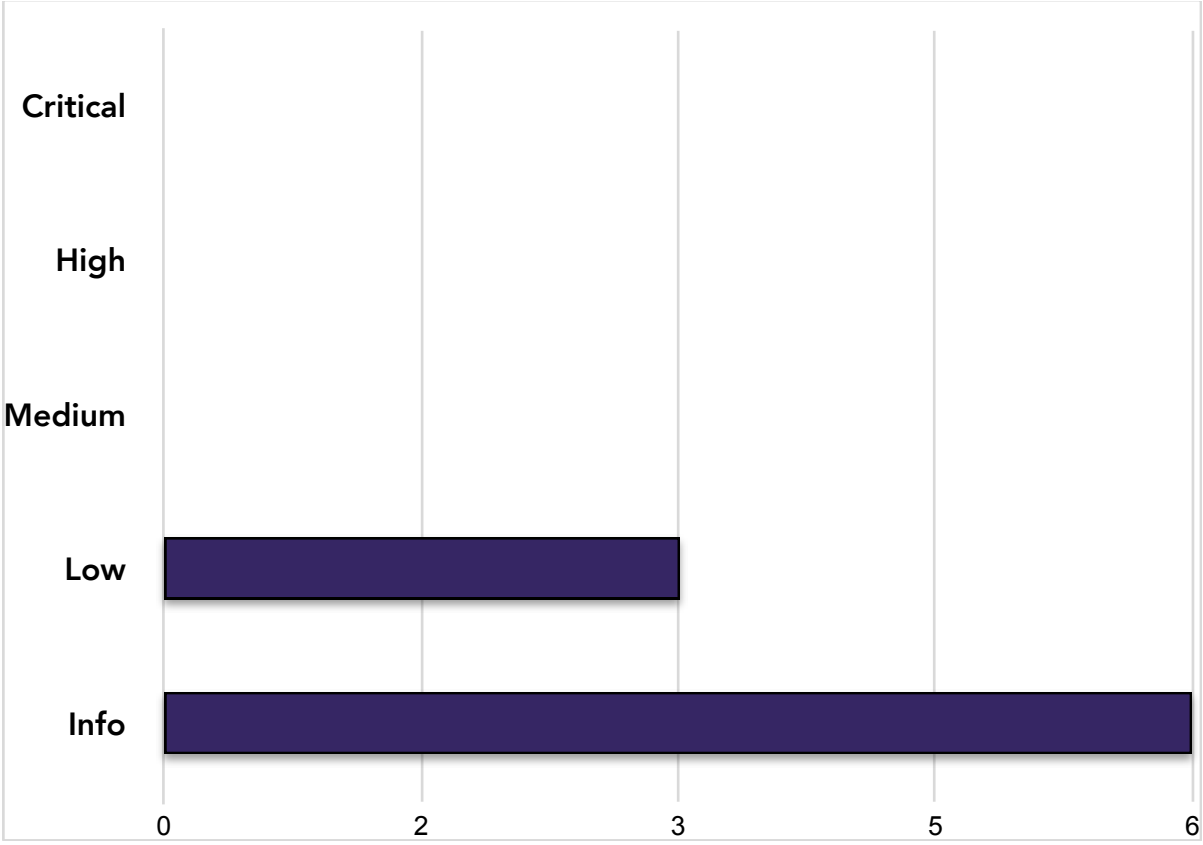

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each finding, including discovery methods, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

#	Severity	Description
KS-ADAXPRO-01	Informational	An issue with the Order datatype
KS-ADAXPRO-02	Informational	Assumed order of the list
KS-ADAXPRO-03	Informational	Hardcoded assumptions may invalidate transactions
KS-ADAXPRO-04	Low	Exception on an empty list
KS-ADAXPRO-05	Low	Partial pattern matching raising an exception
KS-ADAXPRO-06	Low	Assumptions made when filtering lists
KS-ADAXPRO-07	Informational	Possible divide-by-zero exception
KS-ADAXPRO-08	Informational	Assumptions made on contract parameters
KS-ADAXPRO-09	Informational	Risk of lifting unwanted data

Table 2: Findings Overview

Technical analysis

Based on the source code, the validity of the code was verified and confirmed that the intended functionality was implemented correctly and to the extent that the state of the repository allowed.

Based on formal verification, we conclude that the code implements the documented functionality to the extent of the code reviewed.

Technical findings

General observations

We could conclude that the code is well written and solid design during the review. We only found minor flaws that would not have been exploitable in the wild and some informational findings leading to greater clarity of the purpose and execution of the script.

Based on this, we conclude that the contract will perform the intended purpose without any security issues based on the reviewed contract design and implementation.

An issue with the order datatype

Finding ID: KS-ADAXPRO-01
Severity: **Informational**
Status: **Remediated**

Description

The Order datatype doesn't have a corresponding `PlutusTx.makeIsDataIndexed`

Proof of issue

Filename: OnChain.hs
Line number: 47

```
PlutusTx.makeLift ''Order
```

Severity and impact summary

As per the official documentation, records are lifted as indexed data, and having stable sorting of keys, makes them safe for an extension.

Recommendation

If you intend to continue expanding this work, it is essential to consider this.

References

- https://plutus.readthedocs.io/en/latest/plutus/tutorials/basic-validators.html?highlight=makeIsDataIndexed

Assumed order of the list

Finding ID: KS-ADAXPRO-02
Severity: **Informational**
Status: **Remediated**

Description

This predicate checked that the list of recipients includes the validator and whatever was in the first position of the transaction outputs. This means this validator should pass even if we receive a `TxInfo` that includes _only_ the validator address, without the owner's address.

Proof of issue

Filename: OnChain.hs
Line number: 59

```
traceIfFalse "Transaction can only involve script and owner" areRecipientsCorrect &&
```

Severity and impact summary

This is according to the Adax OnChain use case.

Recommendation

None, as this extends the capabilities of the Cardano functionality.

References

- N/A

Hardcoded assumptions may invalidate transactions

Finding ID: KS-ADAXPRO-03
Severity: **Informational**
Status: **Remediated**

Description

Based on the way'amountWithFee` is coded, cases where the fee is zero, will not pass.

Proof of issue

Filename: OnChain.hs
Line number: 60

```
traceIfFalse "Exchange is not valid" (amountWithFee <= qVal)
```

Severity and Impact summary

As Plutus does have only integers, there will be, by design, a zero fee on small amount transactions. This will not be a problem as the user will need to attach a constant ADA fee to the transaction.

Recommendation

To not create problems in the future when this is refactored or revised, we suggest that some comments be added to the code that explains the calculations in detail. This way, it is clear what the intention is.

References

- N/A

Exception on an empty list

Finding ID: KS-ADAXPRO-04
Severity: **Low**
Status: **Remediated**

Description

Using `head` on an empty list will raise an exception.

Proof of issue

Filename: OnChain.hs
Line number: 78

```
pAddress :: Address
pAddress = txOutAddress (head outputsTx)
```

Severity and Impact summary

A transaction will always have output, but in the unlikely case it wouldn't raise an exception, here will fail the script and invalidate the transaction.

Recommendation

Use pattern matching and produce a meaningful error that gives valuable information to debug the invalid transaction.

References

- N/A

Partial pattern matching raising an exception

Finding ID: KS-ADAXPRO-05
Severity: **Low**
Status: **Remediated**

Description

`toPubKeyHash` returns a `Maybe hash`, which results in a partial pattern-matching that will raise an exception

Proof of issue

Filename: OnChain.hs
Line number: 81

```
pHash :: PubKeyHash
pHash = case (toPubKeyHash pAddress) of
   Just hash -> hash
```

Severity and impact summary

Raising an exception here will fail the script and invalidate the transaction.

Recommendation

Use pattern matching and produce a meaningful error that gives valuable information to debug the invalid transaction.

References

- N/A

Assumptions made when filtering lists

Finding ID: KS-ADAXPRO-06
Severity: **Low**
Status: **Remediated**

Description

Using `pAddress` when filtering all the addresses in `txOutAddresses` to validate recipients' correctness requires the list to be correct. As both `txOutAddresses` and `pAddress` are coming from the same `outputsTx` transaction list, we don't have a straightforward way to see what makes a correct list of addresses.

Proof of issue

Filename: OnChain.hs
Line number: 88

```
areRecipientsCorrect :: Bool

areRecipientsCorrect = length (filter (\x -> (x /= validatorAddress) && (x /= pAddress))
txOutAddresses) == 0
```

Severity and impact summary

If the list contains more than the expected values, the list will not be correct.

Recommendation

It is better to handle this failure gracefully with a validation function like this example

```
```haskell
validate [a, b] = (a == scriptOwnerAddr && b == validatorAddr)
                  || (a == validatorAddr && b == scriptOwnerAddr)
validate _ = false
```
```

If the ordering of those elements within the list matters, you may remove the left or right side of that `||`.

References

- N/A

Possible divide-by-zero exception

Finding ID: KS-ADAXPRO-07
Severity: **Informational**
Status: **Remediated**

Description

The `priceDivisor` of order appears in the divisor position, but we cannot guarantee that it will be non-zero, so there's a risk of a divide-by-zero exception.

Proof of issue

Filename: OnChain.hs
Line number: 124

```
amount :: Integer
amount = divideInteger (multiplyInteger bVal (price o)) (priceDivisor o)
```

Severity and impact summary

In a divide-by-zero exception, the script will fail and invalidate the transaction.

Recommendation

If the cost for running extra logic in the script, we would suggest that a more meaningful error message was recorded to debug the failing transaction.

References

- N/A

Assumptions made on contract parameters

Finding ID: KS-ADAXPRO-08
Severity: **Informational**
Status: **Remediated**

Description

The assumption that the `price` and `priceDivisor` relation to the total base value of the script inputs (considering the `baseAssetName` and `baseAssetSymbol`) is correct is not apparent. Using these values without asserting any properties could be an issue when presented with wrong values.

Proof of issue

Filename: OnChain.hs
Line number: 124

```
amount :: Integer
amount = divideInteger (multiplyInteger bVal (price o)) (priceDivisor o)
```

Severity and impact summary

Based on the script running in a controlled environment and that the input parameters are validated in the user input, we can conclude that they will fall within the allowed parameters.

Based on that, this is a contract where the user sets the terms of the contract. Nothing is stopping the user from using their own decided values.

Recommendation

The user should be aware of the impact of using specific values in their contract.

References

- N/A

Risk of lifting unwanted data

Finding ID: KS-ADAXPRO-09
Severity: **Informational**
Status: **Remediated**

Description

When using `liftCode`, you are vulnerable to accidentally lifting data you didn't mean to. In this case, you are already using a typed validator, so you know that the data is well-typed in this context.

Proof of issue

Filename: OnChain.hs
Line number: 136

```
typedValidator :: Order -> Scripts.TypedValidator Adax
typedValidator o = Scripts.mkTypedValidator @Adax
    ($$(PlutusTx.compile [|| mkValidator ||]) `PlutusTx.applyCode` PlutusTx.liftCode o)
    $$(PlutusTx.compile  [|| wrap        ||])
  where
    wrap = Scripts.wrapValidator @BuiltinData @OrderAction
```

Severity and impact summary

In the event of a refactor where the `Order` type was changed, this would leave you exposed.

Recommendation

The Official Cardano docs suggest the usage of `safeLiftCode` where you manually can handle the case of the data not fitting the standard type. This would add extra safety in this area.

References

- N/A