vacuumlabs

Cardano Casino

Audit Report v1

May 31, 2024

Contents

Revision table

Report version	Report name	Date	Report URL
1.0	Main audit	2024-05-31	Full report link

1 Executive summary

THIS REPORT DOES NOT PROVIDE ANY WARRANTY OF QUALITY OR SECURITY OF THE AUDITED CODE and should be understood as a best efforts opinion of Vacuumlabs produced upon reviewing the materials provided to Vacuumlabs. Vacuumlabs can only comment on the issues it discovers and Vacuumlabs does not guarantee discovering all the relevant issues. Vacuumlabs also disclaims all warranties or guarantees in relation to the report to the maximum extent permitted by the applicable law. This report is also subject to the full disclaimer in the appendix of this document, which you should read before reading the report.

Project overview

This project serves as an on-chain part of a Cardano Casino game. It is important to note from the start that the purpose of the smart contracts **is not** to facilitate any game or even randomness that would be run directly on the Cardano chain. It only serves to move bets and winnings between the game and its users and this interaction is fully directed by the off-chain code that was **not** part of this audit.

A *player* can place a bet on the off-chain game, locking a specified amount of Ada in the *Bet* UTxO. This bet is then resolved by the off-chain that determines whether the player won and what are his winnings. The outcome of the game is described as a fraction that should be generated randomly, and this fraction is passed as a redeemer to the validator.

The smart contract is parametrized by a set of intervals that determine the winning multipliers for any interval the generated random number falls into.

If the player loses, the collateral locked in the bet UTxO is forfeited in favor of the *Game* and the player won't receive anything. In other cases, winnings (including the collateral) are sent to the player address that was specified in the datum. Note, that the whole protocol works only with Ada, for both the collateral and the winnings.

Bet resolving transactions need to be signed by a special off-chain `backend key`, so they are fully dependent on the game. Funds are committed as soon as Bets are created and can be unlocked only by the game.

The funds belonging to the game are locked in the *Bank* UTxOs and there can be multiple of them in use in any given transaction. Banks are used to pay out the winners and also collect all the lost bets.

Outside of the bet resolution, bank UTxOs can be changed only in a transaction that is signed by a sufficient number of admin keys that should be fully in control of the game. Such transactions can be used to withdraw any funds that were accumulated. There is no on-chain guarantee that there are sufficient funds to cover any potential winnings at any time.

The whole smart contract relies on one special *Admin* UTxO. This admin UTxO defines the backend key, a set of admin keys as well as fees in % that need to be paid on each bet transaction. The fees are collected for all bets, irrelevant of whether they are winning or losing and are based on the bet amount. The Admin UTxO is used as a reference input in all the previously mentioned transactions. The uniqueness of it is protected by a unique validation token that needs to be present in the UTxO to authenticate its validity. This token was not part of the audit scope and it is up to the client to transparently and verifiably communicate with users about its uniqueness as soon as he deploys the contracts. It needs to be unique, otherwise the whole contract security can be compromised.

Note, that the admin UTxO does not govern the winning intervals, those are baked directly into the smart contract during the compilation and therefore they cannot be changed during the game.

There can be **multiple instances of the protocol**, though. Every instance can use the same smart contract but be parametrized with different intervals. The client expressed an intention to deploy multiple such instances to support different games with different winning intervals and multipliers. Let us repeat that no interval, multiplier or randomness generation mechanism was audited by us.

The admin UTxO can be modified, but only if more than half of the admin keys sign the change transaction. That means that even if some keys are compromised (e.g. the backend key or less than half of the admin keys), those keys can be rotated.

Audit overview

We started the audit at commit `32296358b1b39c4104b22b3c79ea6765f6d821e8` and it lasted from 25 March 2024 to 31 May 2024. The timeframe is inclusive of periods in which we were awaiting the implementation of fixes by the client. We interacted mostly on Slack. The team fixed all issues to our satisfaction. Two issues remain acknowledged, but purely to note that the concern stays on the smart contract level. The client plans to mitigate them off-chain. They do not represent security threats to the system and can be sufficiently mitigated by those means.

The scope of the audit was limited to the smart contract files only. The smart contract are only a supplement to the Cardano Casino game. As mentioned in the summary, they do not generate nor guarantee randomness and they delegate the authority to the backend server which we did not see nor audit.

We did not review nor see any comprehensive tests as part of this audit, and no such tests were included in the repository. As a suggestion for further enhancing the codebase, we recommend integrating tests into the repository and incorporating them into the regular development workflow. We believe that such a step would proactively identify and resolve some issues we found as part of this audit.

We performed a design review followed by a deep manual audit of the code and reported findings along with remediation suggestions to the team in a continuous fashion, allowing the time for a proper remediation that we reviewed afterwards. The design was changed early in the collaboration and the code was rewritten as a result. See more about our methodology in Methodology.

The commit `777d600408e8e87085d0ab506c167b4b71f28f26` represents the final version of the code. The status of any issue in this report reflects its status at that commit. You can see all the files audited and their hashes in Audited files. The smart contract language used is Aiken and the contracts are intended to run on Cardano. To avoid any doubt, we did not audit Aiken itself.

Summary of findings

During the audit, we found and reported: 3 critical, 3 major, 4 medium, 6 minor, and 8 informational findings. All findings were fully resolved except for these 2 that were acknowledged:

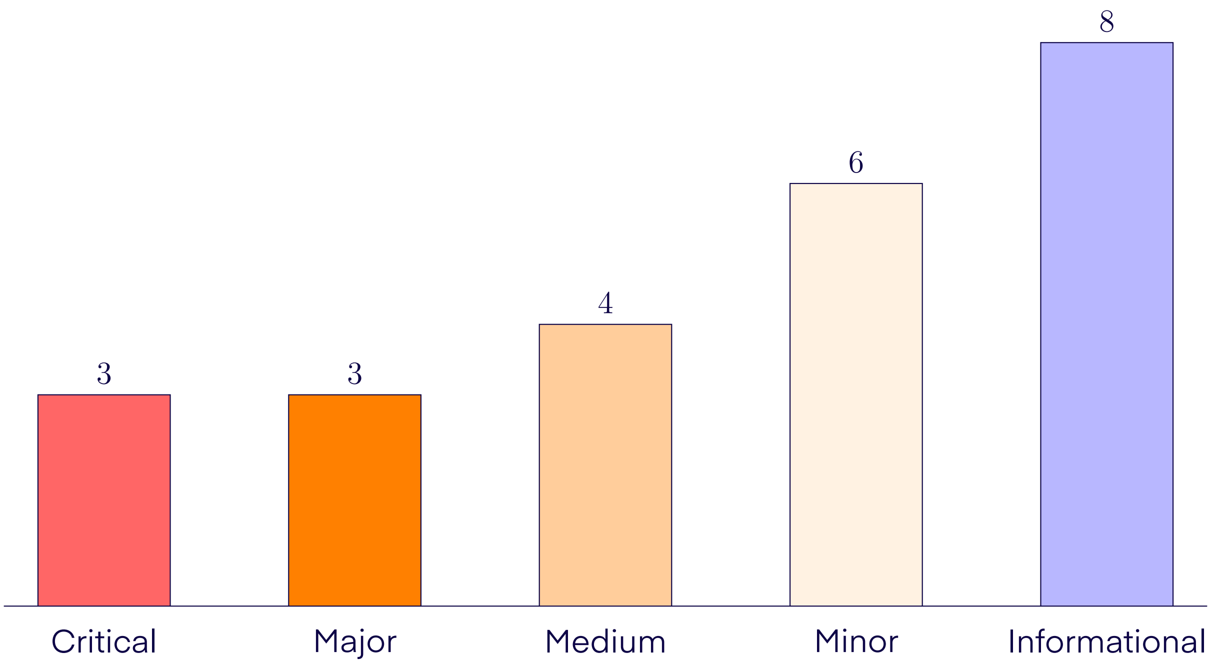
- **CCA-302**: The backend key can single-handedly withdraw all the Bank funds. The key needs some access to the funds. Admin keys can now at least rotate the key in case of misbehavior or loss. We have recommended the client to look at the Bank funds similarly to a hot wallet – leave there only those funds that are needed for day to day operation and periodically move other funds to and from cold storage. This is an off-chain mitigation that limits the exploit surface. However, the risk of draining the Banks by the backend key persists and can not be easily fixed on-chain.
- **CCA-408**: The fee change is hard to handle right. The client needs to be very careful about it, bearing in mind that existing Bets may not have enough funds in them to cover the new fee. It is possible to handle it right solely by proper timing of the changes. As such, it's more of a note than a security threat.

The critical issues were of two basic types. The finding CCA-002 explains an initial problem the design had. It essentially tried to solve the escrow problem with no third party. The design and the code was reworked a lot as part of this issue's resolution.

The findings CCA-001 and CCA-003 arose from code issues. The first one was about an edge case – a possibility to construct a custom-made transaction that includes zero `UpdateRequest` UTxOs which then bypassed important security checks. The third critical finding was about a library function that was used. It worked slightly differently than was assumed which made the validators unsatisfiable. Here, we would like to highlight the importance of maintaining a thorough test coverage that would test the feasibility of the happy case at all times.

The rest of the issues consisted of design improvements, common attacks on Cardano smart contracts such as double satisfaction and dust token attacks, minor edge cases and code style suggestions. There was also a major logical code mistake in CCA-103 and a few issues about the handling of staking credentials in the protocol. Being mindful of the time and client's money and prioritizing the actual value the client gets from our audit, a few code style informational issues remain unreported. They do not pose any security risks.

2 Severity overview



Findings

ID	TITLE	SEVERITY	STATUS
CCA-001	User deposits can be stolen	CRITICAL	RESOLVED
CCA-002	Missing incentives for users to sign a transaction	CRITICAL	RESOLVED
CCA-003	Value checks are not satisfiable	CRITICAL	RESOLVED
CCA-101	`UpdateRequests` can be invalidated	MAJOR	RESOLVED
CCA-102	The backend key can take over the protocol	MAJOR	RESOLVED
CCA-103	Unaccounted Bet funds	MAJOR	RESOLVED

Continued on next page

ID	TITLE	SEVERITY	STATUS
CCA-201	Empty transactions	MEDIUM	RESOLVED
CCA-202	No key redundancy and rotation possibility	MEDIUM	RESOLVED
CCA-203	Non-transparent off-chain computation	MEDIUM	RESOLVED
CCA-204	Double satisfaction on Bet owner's compensation	MEDIUM	RESOLVED
CCA-301	Staking credentials are not handled	MINOR	RESOLVED
CCA-302	Single key can withdraw Bank funds	MINOR	ACKNOWLEDGED
CCA-303	Strict Ada amount comparison	MINOR	RESOLVED
CCA-304	UTxOs can be polluted by dust tokens	MINOR	RESOLVED
CCA-305	UTxOs with a defined staking credential are unprotected	MINOR	RESOLVED
CCA-306	Owner's staking credential can be set arbitrarily	MINOR	RESOLVED
CCA-401	Every UTxO validates the whole transaction	INFORMATIONAL	RESOLVED
CCA-402	Bank inputs and outputs not used in all code branches	INFORMATIONAL	RESOLVED
CCA-403	Fee can be set in whole percents only	INFORMATIONAL	RESOLVED
CCA-404	Duplicated code	INFORMATIONAL	RESOLVED
CCA-405	Unnecessary redeemer in UpdateAdminData	INFORMATIONAL	RESOLVED
CCA-406	Suggest upgrading Aiken and stdlib	INFORMATIONAL	RESOLVED
CCA-407	Unnecessarily strict checks on reference inputs	INFORMATIONAL	RESOLVED

Continued on next page

ID	TITLE	SEVERITY	STATUS
CCA-408	Fee increase could disable active Bets	INFORMATIONAL	ACKNOWLEDGED

CCA-001 User deposits can be stolen

Category	Vulnerable commit	Severity	Status
Logical Issue	`32296358b1`	CRITICAL	RESOLVED

Description

The update part of the validator filters the script inputs looking for `UpdateRequests` to fulfil. However, there can be no requests in the transaction. The check for the user's and Cardano's backend signatures are not enforced in that case. The signatures just help filter out valid requests.

Furthermore, the `all_users_updated` is trivially satisfied, as it only checks that for each valid aggregated request, the particular user is correctly handled.

As a result, anyone can include any user deposit UTxOs in a transaction with zero (valid) `UpdateRequests` and all Ada within can be unlocked and stolen.

Recommendation

First of all, we recommend checking for the Casino's backend signature whenever the `Update` redeemer is used. Additionally, to limit the Casino's ability to unlock any deposits, we recommend enforcing that only deposits relevant to the valid requests are included in the transaction.

Resolution

The issue fix was present at the following commit: `7b475774a8`. The backend's signature is now required and the batching is no longer possible.

CCA-002 Missing incentives for users to sign a transaction

Category	Vulnerable commit	Severity	Status
Design Issue	`32296358b1`	CRITICAL	RESOLVED

Description

All `UpdateRequests` that are executed need to be signed by both the Casino and the impacted users. However, there is no incentive for a user to sign any transaction that would cost him money. He can therefore dodge any loss that would be put onto his account. He can find out about the game result either directly from the transaction he is about to sign or he can inspect the blockchain and find his `UpdateRequests`.

Recommendation

The user needs to commit money to a game before knowing the result. In any other case they can just withdraw the money without applying any loss.

Resolution

The issue fix was present at the following commit: `7b475774a8`. The design was changed to a bit more centralized approach – the users now commit their funds to a specific bet which can not be withdrawn, and that can be resolved only by the backend with its signature. The redeemer for the bet UTxO will contain the result of the game.

CCA-003 Value checks are not satisfiable

Category	Vulnerable commit	Severity	Status
Code Issue	`105f7828a3`	CRITICAL	RESOLVED

Description

There are three occurences of using the `value.policies` function and asserting that the result is a specific list of policies. In no instance does the list include the Ada policy. However, every UTxO contains Ada. That deems all those checks unsatisfiable. That means that the protocol can not operate as it will never validate.

Recommendation

You can either add the Ada policy (`#""`) to the list or use the following construct instead:

```
output.value |> value.without_lovelace |> value.policies == []
```

Resolution

The issue was fixed in the following commit: `777d600408` .

CCA-101 UpdateRequests can be invalidated

Category	Vulnerable commit	Severity	Status
Design Issue	32296358b1	MAJOR	RESOLVED

Description

The update part of the validator filters the script inputs looking for `UpdateRequests` that need to be fulfilled. However, when the correct signature for an `UpdateRequest` is missing this request is just filtered out but can still be part of the transaction.

As a result, anyone can create a transaction with an arbitrary number of `UpdateRequests` that are not correctly signed but the validator will accept such transaction removing all those requests from the chain. Aside from it being a hurdle for the protocol, Ada from those UTxOs could be kept by the attacker as well.

Recommendation

All `UpdateRequests` present in the transaction need to be validated. If a signature is missing for any of them, the whole validation should fail.

Resolution

The issue fix was present at the following commit: `7b475774a8`. The design was changed and the whole concept of `UpdateRequests` was removed.

CCA-102 The backend key can take over the protocol

Category	Vulnerable commit	Severity	Status
Logical Issue	`7b475774a8`	MAJOR	RESOLVED

Description

As all the UTxOs (Bets, Banks, Admin) share the same validator, any redeemer can be used on any of them. Let's assume a single Bet being fulfilled in a `FulfilBet` transaction. The backend can additionally spend the Admin UTxO in that same transaction, using the same `FulfilBet` redeemer on it. Note that it is indeed possible to have the same UTxO in inputs and reference inputs, in this case it would be the Admin UTxO.

Once the backend key spends the Admin UTxO, he can choose to either modify its data to be the sole owner of the protocol or decide to damage and completely halt the protocol. Any such changes are irrevocable.

Recommendation

To avoid using wrong redeemers, you could separate the contract into 3 smart contracts – one for Bets, one for Banks and one for Admin UTxO.

Another way to prevent which would require less refactor, would be to check that there are only Banks and Bets included in a `FulfilBet` transaction.

Resolution

The issue fix was present at the following commit: `b4329c3b80`. The redeemer paths are now distinct and in one transaction only one of them can be fullfilled.

CCA-103 Unaccounted Bet funds

Category	Vulnerable commit	Severity	Status
Logical Issue	49e85b58e4	MAJOR	RESOLVED

Description

The validator uses these two lines to determine the value that should be paid to the user and to the bank, respectively:

```
let user_amount = amount * multiplier
let bank_amount = -user_amount
```

It is then checked that the difference between the outputs and the inputs on the address is at least the computed value.

However, the actual `Bet` input contains amount of Ada plus fees. That is not accounted for in the above calculation.

If the user loses, then setting the $user\_amount = -amount$ and the $bank\_amount = amount$. However, the user balance will be zero which is greater than a negative expected value and thus passes the validator checks. The bank will receive funds that were locked in the `Bet` so its balance is also correct.

However, if the user wins, then the $user\_amount = multiplier * amount$ and $bank\_acount = -multiplier * amount$ for a positive `multiplier` number. In this case, the `user_amount` of funds is transferred from the bank to the user so both balances are correct. However, the funds locked in the `Bet` are unaccounted for and can be sent to any address virtually stealing from the bank.

Recommendation

As the funds inside the `Bet` are locked in the favor of the bank, it should be counted towards the bank balance. With this in place, in case of a win, the user will receive the amount from `Bet` and an additional $(multiplier - 1) * amount$ from the bank UTxOs making all computations correct.

Resolution

The issue fix was present at the following commit: `105f7828a3`. The computation changed to:

```
let user_amount = amount + ( multiplier - 1 ) * amount
let bank_amount = -user_amount + amount
```

The `amount` locked in the Bet is now correctly accounted for.

CCA-201 Empty transactions

Category	Vulnerable commit	Severity	Status
Design Issue	`32296358b1`	MEDIUM	RESOLVED

Description

An attacker can build a transaction with no (valid) `UpdateRequests` while spending and recreating any and all user deposits and bank UTxOs. This way, he can successfully deny the service (DoS) of creating a transaction by the Casino's backend. It is especially troublesome as multiple parties need to sign any such transaction and the UTxO references can not change in the meantime.

Recommendation

Enforce that at least one `UpdateRequest` is executed, the backend signs the transaction and only relevant user deposits are spent. The backend still needs to carefully plan which UTxOs to spend in which transaction.

Resolution

The issue fix was present at the following commit: `7b475774a8`. The design was changed and batching as well as `UpdateRequests` were removed. Additionally, all transactions need to be authorized by at least a single trusted signature.

CCA-202 No key redundancy and rotation possibility

Category	Vulnerable commit	Severity	Status
Design Issue	32296358b1	MEDIUM	RESOLVED

Description

There are 3 different public keys hardcoded in the on-chain codebase. Bank withdrawal is possible whenever two out of two hardcoded keys sign such a transaction. `UpdateRequests` can be carried out if and only if the 3rd key signs that transaction. It is meant as a key used by the backend server.

For any key, it is important to assume two scenarios:

- The private key to a key can be lost. If the backend key is lost, no more `UpdateRequests` could be carried out, requiring people to migrate their deposits and the application to migrate the banks to a v2 with new keys. If any of the other two keys is lost, bank withdrawal can not happen (actually currently it can, see the issue CCA-302).
- The key can be compromised. If the backend key is compromised, any user deposits can be depleted by the attacker. It is not enough to compromise one of the other keys to withdraw Bank funds, after the issue CCA-302 is fixed. However, it might not be possible to withdraw the Bank funds anymore.

Recommendation

We recommend implementing a more robust key management approach. For example, bank withdrawals could be behind a multi-signature scheme consisting of multiple keys requiring less than all of them to unlock funds. The backend key could also be a multi-sig consisting of many keys. The keys should be stored in different locations. We also recommend using a cold storage solution for the bank withdrawal keys.

Furthermore, you could list the keys in a UTxO's datum and require it as a reference input. The script holding it can be a multi-signature validation. That way, you could rotate the keys in case of both a loss and a compromise, as long as you still own the necessary majority.

Resolution

The issue fix was present at the following commit: `7b475774a8`. The design was updated per our suggestions.

CCA-203 Non-transparent off-chain computation

Category	Vulnerable commit	Severity	Status
Logical Issue	`7b475774a8`	MEDIUM	RESOLVED

Description

Bet fulfillment heavily relies on the off-chain. The amount of money that should be rewarded to both the user and the bank is passed to the redeemer and the on-chain simply verifies that these values were sent to appropriate parties.

However, the on-chain doesn't contain any checks to make sure that the passed values are correct. Therefore, an error in the off-chain (which is not audited) which incorrectly sets the `user_amount` to a large number can drain the banks.

Recommendation

It is better to move the computation to the on-chain so the funds are protected by the audited code. It seems, that the only missing parameter for the computation are yields promised to the player in case of a win. If these yields are known at the time of Bet creation they can be a part of the datum.

The `FulfilBet` redeemer will then contain only information on whether the user won or lost his bet.

Resolution

The issue fix was present at the following commit: `e8279b5fd4`. The script is parametrized by a set of intervals that determine the winnings' multiplier based on the random number passed in the redeemer. The whole calculations are therefore done transparently on-chain and even the random numbers generated by the off-chain are accessible on-chain.

CCA-204 Double satisfaction on Bet owner's compensation

Category	Vulnerable commit	Severity	Status
Logical Issue	`49e85b58e4`	MEDIUM	RESOLVED

Description

The validator checks only its own script inputs. If a Bet owner wins, his compensation is sent to his address. The validator checks that it is sent to him and in the correct amount given the randomly generated number provided by the backend. However, there could be multiple scripts expecting a payment to the same address in the same transaction.

As a result, the owner could receive less funds and satisfy all the scripts. The severity is lower as such a transaction would still need to be signed by the backend key. However, as the plans of the project include deploying multiple instances of the contract for different games, it is likely that this finding could be exploited even between different instances of the same platform.

Recommendation

We suggest not allowing any other script inputs in the transaction.

Resolution

The issue fix was present at the following commit: `105f7828a3`. The issue is fixed by allowing only backend's and own script inputs in the transaction.

CCA-301 Staking credentials are not handled

Category	Vulnerable commit	Severity	Status
Design Issue	32296358b1	MINOR	RESOLVED

Description

User deposits as well as Bank UTxOs contain potentially a lot of Ada. That Ada can earn staking rewards. The current on-chain code does not check staking credentials. That means that in any transaction containing user deposits and/or bank UTxOs, the author of the transaction can set the staking credentials of those to any credential he pleases and thus can allocate future staking rewards to himself.

However, realistically, after the CCA-201 issue's recommendation is implemented, a change of the user deposit's staking credential would require their signature. The bank's credential could be updated by the Casino's backend key only.

Recommendation

We recommend implementing the recommendation given in the CCA-201 issue. Afterwards, it is safe to delegate the staking credential handling to the backend, but we recommend being mindful of what credential to pick as anyone can create a UTxO on behalf of anyone.

Resolution

The issue fix was present at the following commit: `49e85b58e4`. The user can choose his staking credentials for both the bet and the compensation. Banks' credentials are handled by the backend key.

CCA-302 Single key can withdraw Bank funds

Category	Vulnerable commit	Severity	Status
Design Issue	32296358b1	MINOR	ACKNOWLEDGED

Description

There is no on-chain validation to distinguish valid `UpdateRequests` from invalid as this responsibility is delegated off-chain. On-chain, a single hardcoded key belonging to the backend server is required to apply `UpdateRequests`. Note, that an `UpdateRequest` can take funds out of any `Bank` UTxO.

As a result, the key alone is sufficient to withdraw all funds out of all `Banks` (e.g. by creating an arbitrary number of "winning" `UpdateRequests`). We consider this unwanted as in the `Bank` withdrawal validation, two different keys are required to both sign the transaction.

Recommendation

We recommend limiting the key's access to the `Bank` funds. Most of the funds could be stored in a cold storage with the key being able to access only a small portion that it needs for day-to-day operation with a manual multi-signature process of moving funds into the hot storage controlled by the key.

Client comment

Bank funds contained within the contract will be kept to a sustainable level that allows the system to work without issues, excess funds will be removed via the admin keys to a cold-storage wallet and any extra funds required to top up the bank will be added if required.

Our comment

`UpdateRequests` are no longer used in the protocol, but the issue persists. If compromised, the backend key can apply any `Bets` including any that were maliciously created and decide for their winnings. It can drain the banks completely.

The client described a good off-chain mitigation of this issue. On-chain, the issue is still present, hence the status of the issue. The impact is mitigated by the suggested solution.

CCA-303 Strict Ada amount comparison

Category	Vulnerable commit	Severity	Status
Design Issue	`32296358b1`	MINOR	RESOLVED

Description

Every UTxO needs to contain at least a minAda amount which is dependent on its size. This puts a small but hard limit on the minimum amount of Ada in any UTxO.

The on-chain validator computes the amount of Ada that needs to be deposited into the deposits and banks UTxOs and require the resulting UTxO to contain exactly the computed amount.

However, if the computed amount is smaller than the required minAda, it is not feasible to create such a UTxO.

Recommendation

We recommend expecting ***at least*** an amount of Ada in output UTxOs instead of checking the *exact* amount.

Resolution

The issue fix was present in the commit `7b475774a8`.

CCA-304 UTxOs can be polluted by dust tokens

Category	Vulnerable commit	Severity	Status
Logical Issue	`7b475774a8`	MINOR	RESOLVED

Description

In any UTxO, just Ada value is checked. That means that in any transaction, other tokens than Ada can be added to the UTxOs. As there is no handling related to other tokens whatsoever, the impact is minimal. However, it can break the transaction loading, decrease the amount of UTxOs that can be put in a single transaction and make random failures. It should not make money unspendable.

Recommendation

We recommend checking that no tokens other than Ada are added to the output script UTxOs.

Resolution

The fix was present in the commit `b4329c3b80`. The banks' outputs are now checked to contain just Ada.

CCA-305 UTxOs with a defined staking credential are unprotected

Category	Vulnerable commit	Severity	Status
Code Issue	`7b475774a8`	MINOR	RESOLVED

Description

The `get_script_inputs` function finds own scripts based on the whole address comparison, finding only those that have undefined staking credential. Suppose some Bets would be created by users directly and those will have staking credentials defined as they do not want to miss on their Ada yield. The scripts do not handle such UTxOs well. They are spendable in any way under any redeemer scenario. As every redeemer scenario enforces at least a single trusted signature, the severity is minor. However, there are no checks for any such UTxOs and for them to be even spendable, transaction would need to be specially constructed.

As a note, Banks undergo the same handling. They are not created by users, though, and so as long as no funds end up (also) there, everything is good. The script makes sure that enough funds end up in the Banks with no staking credential.

Recommendation

When checking script inputs, compare just the payment credential of the address in any such handling. That is inclusive of any staking credential. When checking the outputs, leave the handling as is that is, if you want to enforce particular staking credential.

When checking the outputs, you can check the staking credentials. However, as the protocol design changed, you now do not need to enforce the Banks' staking credentials as only the backend key can sign those transactions and he is probably trusted enough to decide on the staking credential freely.

Resolution

The fix was present in the commit `b4329c3b80`.

CCA-306 Owner’s staking credential can be set arbitrarily

Category	Vulnerable commit	Severity	Status
Logical Issue	`7b475774a8`	MINOR	RESOLVED

Description

An owner of Bet UTxOs is just a public key hash credential. It does not contain the whole address where potential wins should be sent. It is missing the staking credential. That means that it is on the backend key to use owner’s staking credential when it sends wins to the owner. It could be set maliciously to accumulate staking rewards to itself, or it could be set to None. The latter is not handled well by some wallets and could cause the user being unable to see his UTxOs.

Recommendation

We recommend changing the Bet owner field to an address type. Let the owner supply his preferred staking credential. It could be then verified that potential wins are sent exactly to that address. The payment credential can be easily extracted from an address, and so it does not break any of the existing handling.

Beware, though, that there are addresses where you can not create UTxOs at. Make sure you have a policy on how to handle such maliciously constructed Bets, s.a. ignoring them, marking them as lost, or compensating the owner on an address with a different, feasible, staking credential.

Resolution

The issue was fixed in the commit `b4329c3b80` according to our recommendation.

CCA-401 Every UTxO validates the whole transaction

Category	Vulnerable commit	Severity	Status
Optimization	`32296358b1`	INFORMATIONAL	RESOLVED

Description

Both update and withdrawal transactions possibly contain multiple script UTxOs in it. For example, in the update use case, there are multiple `UpdateRequests`, some `Deposits` and some `Bank` UTxOs. For each of those UTxOs, the validator is run. However, the validation is exactly the same in each UTxO as the validator looks at the whole context, aggregating all the data. This is an inefficiency that enlarges the fees paid and limits the number of UTxOs that can be validated within a single transaction.

Recommendation

The validation could be run just once for the whole transaction. This can be achieved by the use of transaction tokens validating the transaction in its minting policy (read more here) or by limiting the validation to *only* the first script input.

Resolution

The updated design implemented in the commit `7b475774a8` does not support fulfilling multiple bets in a single transaction. That makes the issue less important.

Ultimately, the optimalization was implemented in the commit `105f7828a3`. The issue is fully resolved now.

CCA-402 Bank inputs and outputs not used in all code branches

Category	Vulnerable commit	Severity	Status
Code Style	32296358b1	INFORMATIONAL	RESOLVED

Description

In the on-chain codebase, the following variables are computed before any validation happens: `bank_inputs`, `bank_outputs`, `bank_input_ada`, `bank_output_ada`.

They are not used in the `Withdraw` transactions, though. As such, they don't need to be always computed, which can save transaction fees and result in a bigger number of batched UTxOs during the `Withdraw`.

Recommendation

We recommend moving the code to the `Update` redeemer code validation part.

Resolution

The issue fix was present at the following commit: `7b475774a8`. The design was updated and code handling this part was localized to the code branch that needs it.

CCA-403 Fee can be set in whole percents only

Category	Vulnerable commit	Severity	Status
Code Issue	`7b475774a8`	INFORMATIONAL	RESOLVED

Description

The fee protocol parameter is denominated in whole percents. As such, modifying the protocol to take, e.g. 2.5% is impossible.

Recommendation

To be more future-proof, we recommend updating the basis to allow for more granularity, e.g. 1 basis point which is 1 / 100th of a single %. We further recommend adding a comment to the data type explaining what the fee number represents, namely its denomination.

Resolution

The issue fix was present at the following commit: `49e85b58e4`. The granularity is now 1 basis point.

CCA-404 Duplicated code

Category	Vulnerable commit	Severity	Status
Code Style	`7b475774a8`	INFORMATIONAL	RESOLVED

Description

The code performs the same or similar computations in several places using the same lines of code. In the `UpdateAdminData` branch, the `admin_inputs` are extracted with a code that is similar to the `get_admin_data` function. The only difference is that the `get_-admin_data` returns the datum. This function can be modified to return the actual input and another function can retrieve the datum.

Also, both the `UpdateAdminData` and `WithdrawBank` code branches need to count the number of admin keys that signed the transaction which can be delegated to a common function. Such duplicated code makes the code unnecessarily bigger and prone to errors when one part is modified and the other is not.

Recommendation

We recommend refactoring the code to remove the duplicated code and replacing it with smaller dedicated functions.

Resolution

The duplicated code was removed in the commit `b4329c3b80` according to our recommendation.

CCA-405 Unnecessary redeemer in UpdateAdminData

Category	Vulnerable commit	Severity	Status
Logical Issue	`7b475774a8`	INFORMATIONAL	RESOLVED

Description

The `UpdateAdminData` branch checks that the code was signed by a sufficient number of admin keys and then allows for the admin UTxO change. It then checks that this UTxO was changed according to the redeemer.

However, the transaction creator already got approval to change the UTxO, and the final form is present in the signed transaction. The redeemer is therefore unnecessary as it will simply mirror what is already written in the transaction.

Recommendation

When changing the admin UTxO it is sufficient to check that the transaction was correctly signed and an additional check that will verify that the new admin UTxO with a correct validation token was created.

Resolution

The issue was resolved in the commit `b4329c3b80` according to our recommendation.

CCA-406 Suggest upgrading Aiken and stdlib

Category	Vulnerable commit	Severity	Status
Code Issue	`7b475774a8`	INFORMATIONAL	RESOLVED

Description

The upgrading is not crucial for the security as there was no relevant vulnerability fixed in the newer versions, but newer Aiken versions fail compiling these contracts. Additionally, there have been useful changes in the new Aiken – namely, it now writes the compiler version into the `plutus.json` compiled artefacts. This means that it can be verified that the code really compiles into the artefacts more easily.

Recommendation

We recommend upgrading Aiken and the stdlib library.

Resolution

The issue fix was present at the following commit: `b4329c3b80`.

CCA-407 Unnecessarily strict checks on reference inputs

Category	Vulnerable commit	Severity	Status
Code Issue	`49e85b58e4`	INFORMATIONAL	RESOLVED

Description

The `get_admin_data` function tries to obtain the admin data from the admin UTxO passed as a reference input. However, it expects that all reference inputs contain an `InlineDatum` and fails if that is not the case.

It is not a problem as long as e.g. the admin UTxO is the only reference input. However, it unnecessarily limits the transaction.

Recommendation

As additional reference inputs do not cause any problems, the `get_admin_data` function can first filter all the reference inputs with the admin script address and only then look for the correct UTxO by parsing the datum and checking the token.

Resolution

The issue fix was present at the following commit: `105f7828a3`. The issue is fixed according to our recommendation.

CCA-408 Fee increase could disable active Bets

Category	Vulnerable commit	Severity	Status
Logical Issue	49e85b58e4	INFORMATIONAL	ACKNOWLEDGED

Description

The off-chain deems correct only UTxOs with a sufficient amount of funds. This amount should be enough to cover the Bet amount and fees that need to be paid during the bet resolution.

The fee percentage is set by the admin UTxO which can change. It is possible that a valid Bet UTxO is created, but after a fee increase, this UTxO is deemed incorrect by the off-chain. Moreover, to pass it through the validator the difference in the fee needs to be paid.

Recommendation

We recommend solving this issue through the off-chain. For example, you could set a time since when the fee is effective, create new bets with the new fee after that, resolve all the old bets and execute the on-chain fee parameter change just then.

Client comment

All active bets will be fulfilled before a fee change occurs in order to ensure no bets are stuck in the script due to the incorrect fee.

Our comment

The issue was acknowledged and the client had agreed to use the recommended off-chain solution.

A Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the agreement between VacuumLabs Bohemia s.r.o. (VACUUMLABS) and Cardano Casino (CLIENT) (the AGREEMENT), or the scope of services, and terms and conditions provided to the Client in connection with the Agreement, and shall be used only subject to and to the extent permitted by such terms and conditions. THIS REPORT MAY NOT BE TRANSMITTED, DISCLOSED, REFERRED TO, MODIFIED BY, OR RELIED UPON BY ANY PERSON FOR ANY PURPOSES WITHOUT VACUUMLABS'S PRIOR WRITTEN CONSENT.

THIS REPORT IS NOT, NOR SHOULD BE CONSIDERED, AN ENDORSEMENT, APPROVAL OR DISAPPROVAL of any particular project, team, code, technology, asset or anything else. This report is not, nor should be considered, an indication of the economics or value of any technology, product or asset created by any team or project that contracts Vacuumlabs to perform a smart contract assessment. THIS REPORT DOES NOT PROVIDE ANY WARRANTY OR GUARANTEE REGARDING THE QUALITY OR NATURE OF THE TECHNOLOGY ANALYSED, nor does it provide any indication of the technology's proprietors, business, business model or legal compliance.

To the fullest extent permitted by law, VACUUMLABS DISCLAIMS ALL WARRANTIES, EXPRESSED OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, AND THE RELATED SERVICES AND PRODUCTS AND YOUR USE THEREOF, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. This report is provided on an as-is, where-is, and as-available basis. Vacuumlabs does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by Client or any third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services, assets and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and VACUUMLABS WILL NOT BE A PARTY TO OR IN ANY WAY BE RESPONSIBLE FOR MONITORING ANY TRANSACTION BETWEEN YOU AND CLIENT AND/OR ANY THIRD-PARTY PROVIDERS OF PRODUCTS OR SERVICES.

THIS REPORT SHOULD NOT BE USED IN ANY WAY BY ANYONE TO MAKE DECISIONS AROUND INVESTMENT OR INVOLVEMENT WITH ANY PARTICULAR PROJECT, services or assets, especially not to make decisions to buy or sell any assets or products. This report provides general information and is not tailored to anyone's specific situation, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or other advice.

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Vacuumlabs prepared this report as an informational exercise documenting the due diligence involved in the course of development of the Client's smart contract only, and THIS REPORT MAKES NO CLAIMS OR GUARANTEES CONCERNING THE SMART CONTRACT'S OPERATION ON DEPLOYMENT OR POST-DEPLOYMENT. This report provides no opinion or guarantee on the security of the code, smart contracts, project, the related assets or anything else at the time of deployment or post deployment. Smart contracts can be invoked by anyone on the internet and as such carry substantial risk. VACUUMLABS HAS NO DUTY TO MONITOR CLIENT'S OPERATION OF THE PROJECT AND UPDATE THE REPORT ACCORDINGLY.

THE INFORMATION CONTAINED IN THIS REPORT MAY NOT BE COMPLETE NOR INCLUSIVE OF ALL VULNERABILITIES. This report is not comprehensive in scope, it excludes a number of components critical to the correct operation of this system. You agree that your access to and/or use of, including but not limited to, any associated services, products, protocols, platforms, content, assets, and materials will be at your sole risk. On its own, it cannot be considered a sufficient assessment of the correctness of the code or any technology. This report represents an extensive assessing process intending to help Client increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology, however blockchain technology and cryptographic assets present a high level of ongoing risk, including but not limited to unknown risks and flaws.

While Vacuumlabs has conducted an analysis to the best of its ability, it is Vacuumlabs's recommendation to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring and/or other auditing and monitoring in line with the industry best practice. The possibility of human error in the manual review process is highly real, and Vacuumlabs recommends seeking multiple independent opinions on any claims which impact any functioning of the code, project, smart contracts, systems, technology or involvement of any funds or assets. VACUUMLABS'S POSITION IS THAT EACH COMPANY AND INDIVIDUAL ARE RESPONSIBLE FOR THEIR OWN DUE DILIGENCE AND CONTINUOUS SECURITY.

B Audited files

The files and their hashes reflect the final state at commit `777d600408e8e87085d0ab506c167b4b71f28f26` after all the fixes have been implemented.

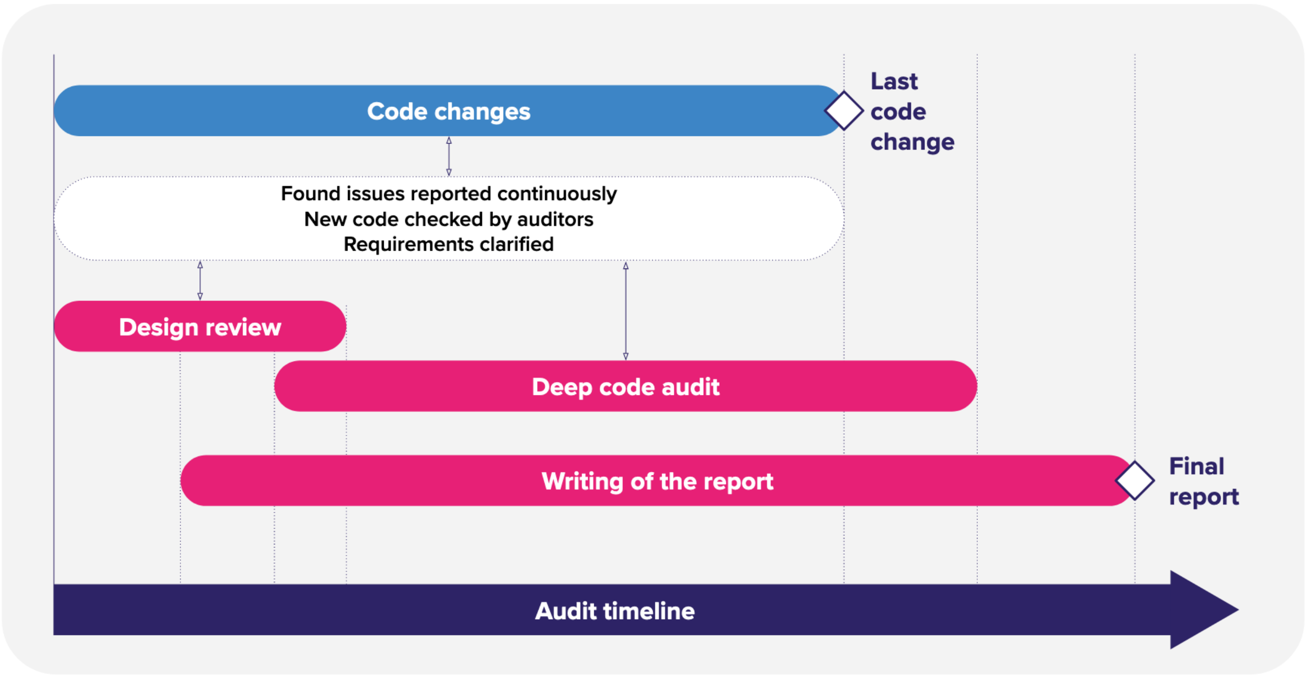
SHA256 hash	Filename
54a93...d5c6a	lib/cardanocasino/types.ak
3289a...7d802	validators/bets.ak

Please note that we did not audit the legitimate token policy as it was not part of the codebase. The client aims to use a simple time-bound native script for that. We expect that the client will transparently communicate with the users about a proper setup of the token and the following deployment of the scripts.

C Methodology

Vacuumlabs' agile methodology for performing security audits consists of several key phases:

1. Design reviews form the initial stage of our audits. The goal of the design review is to find larger issues which result in large changes to the code fast.
2. During the deep code audit, we verify the correctness of the given code and scrutinize it for potential vulnerabilities. We also verify the client's fixes for all discovered vulnerabilities. We provide our clients with status reports on a continuous basis providing them a clear up-to-date status of all the issues found so far.
3. We conclude the audit by handing over a final audit report which contains descriptions and resolutions for all the identified vulnerabilities.



Throughout our entire audit process, we report issues as soon as they are found and verified. We communicate with the client for the duration of the whole audit. During our audits, we check several key properties of the code:

- Vulnerabilities in the code
- Adherence of the code to the documented business logic
- Potential issues in the design that are not vulnerabilities
- Code quality

During our manual audits, we focus on several types of attacks, including but not limited to:

1. Double satisfaction
2. Theft of funds
3. Violation of business requirements
4. Token uniqueness attacks
5. Faking timestamps
6. Locking funds indefinitely
7. Denial of service
8. Unauthorized minting
9. Loss of staking rewards

D Issue classification

Severity levels

The following table explains the different severities.

Severity	Impact
CRITICAL	Theft of user funds, permanent freezing of funds, protocol insolvency, etc.
MAJOR	Theft of unclaimed yield, permanent freezing of unclaimed yield, temporary freezing of funds, etc.
MEDIUM	Smart contract unable to operate, partial theft of funds/yield, etc.
MINOR	Contract fails to deliver promised returns, but does not lose user funds.
INFORMATIONAL	Best practices, code style, readability, documentation, etc.

Resolution status

The following table explains the different resolution statuses.

Resolution status	Description
RESOLVED	Fix applied.
PARTIALLY RESOLVED	Fix applied partially.
ACKNOWLEDGED	Acknowledged by the project to be fixed later or out of scope.
PENDING	Still waiting for a fix or an official response.

Categories of issues

The following table explains the different categories of issues.

Category	Description
Design Issue	High-level issues in the design. Often large in scope, requiring changes to the design or massive code changes to fix.
Logical Issue	Medium-sized issues, often in between the design and the implementation. The changes required in the design should be small-scaled (e.g. clarifying details), but they can affect the code significantly.
Code Issue	Small in size, fixable solely through the implementation. This category covers all sorts of bugs, deviations from specification, etc.
Code Style	Parts of the code that work properly but are possible sources of later issues (e.g. inconsistent naming, dead code).
Documentation	Small issues that relate to any part of the documentation (design specification, code documentation, or other audited documents). This category does not cover faulty design.
Optimization	Ideas on how to increase performance or decrease costs.

E Report revisions

This appendix contains the changelog of this report. Please note that the versions of the reports used here do not correspond with the audited application versions.

v1.0: Main audit

Revision date:	2024-05-31
Final commit:	`777d600408e8e87085d0ab506c167b4b71f28f26`

We conducted the audit of the main application. To see the files audited, see Executive Summary.

Full report for this revision can be found at url.

F About us

Vacuumlabs has been building crypto projects since the early days.

- We helped create WingRiders, currently the second largest decentralized exchange on Cardano (based on TVL).
- We are behind the popular AdaLite wallet. It was later improved into a multichain wallet NuFi.
- We built the Cardano applications for the hardware wallets Ledger and Trezor.
- We built the first version of the cutting-edge decentralized NFT marketplace Jam On Bread on Cardano with truly unique features and superior speed of both the interface and transactions.

Our auditing team is chosen from the best.

- Talent from esteemed Cardano projects: WingRiders and NuFi.
- Rich experience across Google, traditional finance, trading and ethical hacking.
- Award-winning programmers from ACM ICPC, TopCoder and International Olympiad in Informatics.
- Driven by passion for program correctness, security, game theory and the blockchain technology.

We are a trusted Cardano ecosystem development partner

Contact us:

audit@vacuumlabs.com