



AADA

Audit Report, v0.9

August 29, 2022

Disclaimer: The audit makes no statements or warranties on the security of the code. On its own, it can not be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring.

Executive summary

Project overview

The project offers peer-to-peer decentralized lending. A person interested in borrowing assets (Borrower) can create a loan request with information about the loan including its duration and interest. The created request carries locked collateral to guarantee the loan repayment. Borrower gets a bond NFT (Borrower NFT) in exchange for creating the request. It can later be used to cancel the request, repay the loan and unlock the collateral.

Someone else can fulfill the request (Lender) by sending the requested loan amount to the Borrower. They get back a bond NFT (Lender NFT) proving that they lent the assets. It can be later used to collect the repayment along with interest.

Both bond NFTs – both Borrower NFT and Lender NFT – can be sold on 3rd party marketplaces or otherwise. Responsibilities of the respective party are transferred alongside the NFT ownership.

Borrower NFT holder can pay the loan and interest back before the agreed loan duration expires (deadline).

If that is the case, interest is paid proportionally to the time that has passed. Full interest amount is paid after the full agreed loan duration has passed, 50% of the interest is paid after half of the loan duration, etc.

If the loan is not repaid on time, the Lender NFT holder can liquidate and claim the whole collateral.

Collateral value should be worth more than the loan and interest at all times. If it drops significantly in value, Lender NFT holder can initiate a liquidation of the collateral. In this case, the liquidation needs to be accompanied by a liquidation token minting. The liquidation policy is specified by the Borrower in the loan request. In the default policy provided, three oracle nodes need to sign the liquidation off.

Other important details:

- Staking part of the smart contract addresses is a fixed contract parameter.
- We emphasize that the liquidation policy is chosen by the Borrower in the loan request. It is up to the Lender to check that he trusts it before lending money. In theory, the protocol is expandable by new more robust liquidation policies in the future. The

security of the protocol depends on the security of the liquidation policy used. We reviewed only the provided default three oracle nodes sign-off policy.

- The language used is Plutus V1. Upgrading the scripts to Plutus V2 slightly changes the security model. As a result, further review is required before upgrading the existing scripts.

Methodology

The first phase of our audit collaboration was a design review. We reviewed the high-level design and suggested improvements and simplifications. The major design improvement was the removal of two unnecessary tokens whose functionality was replaced by carrying the information in datums. The simplified design led to significant code reduction. Note that there might have been vulnerabilities present before the design simplification, i.e. before commit `77ef49bf35c45d5a8a442533f4562ea016ea12cf`, that have been fixed by the design simplification.

After the design review, we conducted a deeper manual audit of the code and reported findings to the team in two batches. We clarified the business requirements further as detailed documentation was not available and made sure the code adheres to them.

Our manual process focused on, including, but not limited to, the following attacks:

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

The audit lasted from 26 July 2022 to 29 August 2022. We interacted on Discord and gave feedback in GitHub pull requests. The team fixed most of the issues and acknowledged the rest. The acknowledged ones are marked in the report accordingly.



































Files audited

The files and their hashes reflect the final state at commit

b098e45206b4b1803cf2496cbf99011ff72b6162 after all the fixes have been implemented.

SHA256 hash	Filename
ee6b1...49129	src/AadaNft.hs
bcaa6...bf411	src/Collateral.hs
b4755...2032f	src/Common/Utils.hs
de723...01e93	src/Interest.hs
c6008...68df3	src/Liquidation.hs
82eb5...eb48a	src/OracleNft.hs
4c528...a4a30	src/Request.hs

Findings

AADA-001	Borrower/Lender tokens do not have to be NFTs	 Critical	Resolved 
AADA-002	Double Satisfaction – Collateral	 Critical	Resolved 
AADA-003	Double Satisfaction – Loan	 Critical	Resolved 
AADA-004	Resource DoS attack	 Critical	Resolved 
AADA-005	Lender can take collateral when providing loan	 Critical	Resolved 
AADA-006	Lender can take the whole collateral in liquidation after deadline	 Critical	Acknowledged
AADA-101	Borrower may repay loan without paying interest	 Major	Resolved 
AADA-102	Lender can collect interest immediately	 Major	Resolved 
AADA-103	Staking key can be freely modified for collateral	 Major	Resolved 
AADA-104	Burning of oracle tokens can liquidate	 Major	Resolved 
AADA-105	Structure of the liquidation transaction is not enforced	 Major	Acknowledged
AADA-201	Repaying the loan near deadline can be DoS-ed	 Medium	Acknowledged
AADA-202	Losing partial interest amount precision	 Medium	Resolved 
AADA-203	Incomplete documentation	 Medium	Pending
AADA-301	Loss of Min-ADA	 Minor	Pending
AADA-302	Rounding of interest amount favors borrower	 Minor	Pending
AADA-303	Liquidation checks too lenient	 Minor	Resolved 
AADA-304	Default liquidation policy relies on all 3 oracles working	 Minor	Acknowledged
AADA-305	Constant protocol staking key rewards are capped	 Minor	Acknowledged
AADA-401	Dead Code	 Informational	Resolved 
AADA-402	Inverted percentage calculation	 Informational	Resolved 

AADA-403	Possible minting of oracle tokens with different token name	<input type="checkbox"/> Informational	Resolved ✓
AADA-404	Naming inconsistencies and improvements	<input type="checkbox"/> Informational	Resolved ✓
AADA-405	Duplicated logic	<input type="checkbox"/> Informational	Pending

AADA-001: Borrower/Lender tokens do not have to be NFTs

Severity: ■ Critical

Status: Resolved ✓

Description

The code requiring the minting of the Borrower and Lender NFTs doesn't check the currency symbol. That means that the Borrower and the Lender can both use a regular token instead of an actual NFT.

A malicious Borrower/Lender could sell their bond token and mint another of the same token. The buyer would mistakenly think that they are the sole owner of the supposed NFT and therefore have sole rights to the collateral/interest/loan. However, the malicious Borrower/Lender can use their second token to collect the collateral/interest/loan instead.

Recommendation

We suggest forcing the usage of an easily verifiable minting policy that is guaranteed to be an NFT policy. Ideally, the currency symbol would be constant.

Resolution

The usage of an audited NFT minting policy with a constant currency symbol is used for both Borrower and Lender NFTs. It is also enforced on the protocol level. There are different currency symbols for Lender NFTs vs. Borrower NFTs.

AADA-002: Double Satisfaction – Collateral

Severity:  Critical

Status: Resolved 

Description

Lender can batch multiple loan requests together in a single transaction and only lock the maximum collateral amount of all the collateral amounts locked in the spent loan requests in a Collateral UTxO. It is possible to steal the remaining collateral value.

Recommendation

We suggest disallowing batching.

AADA-003: Double Satisfaction – Loan

Severity: ■ Critical

Status: Resolved ✓

Description

Lender can batch multiple loan requests in a single transaction. If multiple of those specify the same Borrower, the Lender may provide loan only for the maximum of the wanted amounts (per loan currency). This effectively deprives the Borrower of money. However, all the loans would need to be repaid or the locked collaterals could be liquidated.

Another attack to achieve the same result is to mix a loan request of the same Borrower with other protocol's smart contracts. Lender can include a loan request with any other smart contract that does not forbid it and that expects the Borrower getting paid. An example could be an NFT marketplace with the Borrower listing an item for sale. Lender could then pay only the highest expected amount instead of the sum and satisfy all the scripts.

Recommendation

We suggest making sure the loan request is the only smart contract among the transaction inputs.

AADA-004: Resource DoS attack

Severity: ■ Critical

Status: Resolved ✓

Description

Borrower NFT holder can add additional useless tokens in the Return transaction, making the output UTxO too big, so that the Lender NFT holder has trouble doing a Claim transaction. The useless tokens increase the execution units needed for a successful Claim transaction, effectively preventing the Lender NFT holder from Claiming.

Similarly, Lender NFT holder can add additional useless tokens making the Return transaction impossible to make for a Borrower NFT holder, but the Lender NFT holder might still be able to go ahead with Liquidation (as that is easier on the execution units). The motivation behind the attack is obvious: The Lender NFT holder can therefore liquidate once the deadline passes and claim the whole collateral whose value is bigger than that of the loan.

Recommendation

Check that the output UTxOs do not contain any more tokens than those necessary (loan, collateral, interest, min-ADA).

AADA-005: Lender can take collateral when providing loan

Severity: ■ Critical

Status: Resolved ✓

Description

The check in `containsRequiredCollateralAmount` in `Request.hs` file uses the opposite value comparison.

It checks that the collateral in the input (Request script UTxO) is more or the same compared to the ongoing Collateral script UTxO. This means that the Lender can take the whole collateral when providing the loan.

Recommendation

Reverse the comparison and check that the ongoing locked collateral is bigger or equal to the value locked before.

We also suggest comparing the ongoing locked collateral to the collateral value noted in the Request datum instead of the currently locked value.

AADA-006: Lender can take the whole collateral in liquidation after deadline

Severity: ■ Critical

Status: Acknowledged

Description

There are two types of liquidations. Both require the current lender NFT holder to initiate it, but depending on whether the loan deadline has already passed or not, the oracle nodes may not be needed to sign the transaction. This leads to an inconsistency between the two liquidations.

The liquidation with oracles trusts the oracle nodes to ensure a proper transaction structure. This among other things means that the left-over collateral is locked into the Liquidation script that can be claimed by the Borrower NFT holder later on.

The liquidation done purely by the Lender does not enforce any liquidation transaction structure. The lender NFT holder is free to take the whole collateral once the deadline has passed.

Recommendation

We would suggest adding business checks about the liquidation transaction structure directly into the validator.

Furthermore, if the left-over collateral is supposed to go to the borrower NFT holder, we suggest requiring oracle nodes to provide the exchange rate in the transaction.

Resolution

According to the client, this will be improved in the next version when there will be a better oracle design. Right now this option exists as a safe-guard in case the oracle-based liquidation becomes unusable.

AADA-101: Borrower may repay loan without paying interest

Severity: ■ Major

Status: Resolved ✓

Description

The partial interest calculation is proportional to the time that has elapsed since the loan was provided. It takes `interestPayDate` which is a redeemer to be the current time approximation. However, the time stamp may be arbitrarily chosen and thus it may be set such that the partial interest computation would compute zero interest. As a result, the borrower NFT holder can repay the loan without paying interest.

The issue lies in the insufficient time validation checking that the `interestPayDate` is a good enough time approximation for the use case.

Recommendation

Use the end of the transaction range as `interestPayDate`. We know that this time will be in the future and thus the borrower will have to pay enough interest.

AADA-102: Lender can collect interest immediately

Severity: ■ Major

Status: Resolved ✓

Description

The partial interest calculation is proportional to the time that has elapsed since the loan was provided. Lender could set the start of the loan to any point in the past, so when Borrower repays the loan he can be forced to pay the whole interest even though not enough time has passed for that.

The issue lies in the insufficient time validation checking that the `interestPayDate` is a good enough time approximation for the use case.

Recommendation

Use the end of the transaction range as `lendDate`. We know that this time will be in the future and thus the lender cannot move it into the past.

AADA-103: Staking key can be freely modified for collateral

Severity:  Major

Status: Resolved 

Description

Collateral is locked into smart contracts by various actors. Each actor can modify the staking part of the address, so e.g. Lender can profit from staking of collateral during loan duration.

Recommendation

The business specification says that collateral should be staked in a stake pool chosen by the AADA project. The staking part of each address should be checked that it's equal to the chosen stake pool.

AADA-104: Burning of oracle tokens can liquidate

Severity:  Major

Status: Resolved 

Description

In the liquidation transaction, `checkForLiquidationNft` does not validate that the oracle token was minted. It can be burned and still be able to liquidate any loan.

What's more, burning of the default provided oracle token does not check the oracle node signatures. It validates that the token resides at `dest` address which is the policy parameter. However, there's no specification on who owns the destination address. It all boils down to who owns the address. If nobody, the check would drain min-ADA amount for every liquidation. If it's a person, they can liquidate any collateral making it critical from the security perspective.

Recommendation

First and foremost, we recommend checking that oracle tokens are minted and not burned in `checkForLiquidationNft`. Taking into account that the provided default oracle policy may not be the only one being used, we believe it's even more important to make this clear as future policies can easily make the same mistake.

Once the above recommendation is in place, it's not necessary to check the destination of oracle tokens in the default oracle policy. Therefore, we suggest rewriting and simplifying the policy to check for signatures only.

AADA-105: Structure of the liquidation transaction is not enforced

Severity: ■ Major

Status: Acknowledged

Description

The structure of the liquidation transaction is not enforced in the validator. This includes both types of liquidations – the liquidation with oracle and the liquidation without oracle.

We consider this insufficient and not a good practice as the liquidation mechanism is critical to the security and by not checking the structure on-chain, the security relies on an off-chain closed-source code running by the oracles. This decision adds many more security assumptions on the proper deployment and security practices, bug-free off-chain code, etc.

Recommendation

Enforce proper structure of the liquidation transaction in the validator instead of relying on the off-chain code.

Resolution

According to the client, this will be fixed in the next version together with a better oracle design.

AADA-201: Repaying the loan near deadline can be DoS-ed

Severity: ■ Medium

Status: Acknowledged

Description

The borrower NFT holder can lose the whole collateral if he does not manage to repay the loan on time. If he chooses to repay it near the deadline, he may not be able to manage it because of the specifics of the chain (e.g. the network can be congested).

The lender NFT holder can spam the chain, making the borrower NFT holder's transaction not go through. He can then make a transaction liquidating the whole collateral and can with some negligible probability make it first. The borrower can thus lose a lot, even though he wanted to repay the loan in full on time.

Recommendation

We recommend minimizing the difference between repaying the loan and liquidating the collateral by the lender NFT holder only.

A way to do this could in our opinion be restricting the lender NFT holder to take only a part of the collateral upon liquidation. The part would be decided fairly based on the exchange rate provided by oracle nodes.

Resolution

According to the client, this will be fixed in the next version together with a better oracle design.

AADA-202: Losing partial interest amount precision

Severity: ■ Medium

Status: Resolved ✓

Description

The partial interest computation goes through a few steps, losing precision on the way. Plutus uses `Integer` types and thus any division is flooring the (intermediary) result.

Computing the partial interest first computes the percentage of the time elapsed and rounds it down to a single percentage. Then it multiplies it by the actual interest amount.

As a result, the partial interest is unnecessarily rounded into approximately 1-percent precision making the computation imprecise with the lender NFT holder losing on this. The bigger the interest amount, the bigger the loss.

Recommendation

We suggest abandoning the intermediary rounding altogether. We suggest not computing the percentage and instead compute the partial interest amount directly with divisions being the last steps in the equation.

AADA-203: Incomplete documentation

Severity: ■ Medium

Status: Pending

Description

There are places where documentation is lacking. Very important business requirements are lacking. The user right now needs to read code in order to figure out the following.

- Which address gets staking rewards for collateral
- That interest doesn't have to be present until it's repaid by Borrower, as the documentation in Gitbook is wrong.

There is no code for liquidation and documentation does not contain the exact rules for liquidation, so the datum parameters 'collateralFactor' and 'liquidationCommission' have unclear meaning.

There are more examples, but these three are the most significant.

Recommendation

We recommend first and foremost documenting business requirements, so that users know exactly what to expect from the application.

It's also important to document how these business requirements translate to code, so that the implementation can be further checked by interested users.

Resolution

We reported the issue but as far as we know, it hasn't been addressed.

AADA-301: Loss of Min-ADA

Severity:  Minor

Status: **Pending**

Description

Min-ADA from Borrower that is present in the Request UTxO can be ultimately taken away by the Lender in the case when all the tokens (Collateral, Interest, Loan) are not ADA and Borrower needs to add min-ADA value to the Request UTxO. It also needs to be present in all the other script UTxOs and ultimately can be claimed by the Lender upon a successful Claim transaction. This can be an inconsistency between the expectations of the borrower and reality. However, we are talking about 2 ADA, so it's a low severity.

Recommendation

Mitigation is again more of a business decision. It can be either clearly communicated to the end users or it can be catered on the smart contract side (which could get messy, though).

Resolution

We reported the issue but haven't heard back about it.

AADA-302: Rounding of interest amount favors borrower

Severity: ■ Minor

Status: Pending

Description

The partial interest amount computation works on Integer types and uses division several times. Every division floors the number. The result of the computation is the amount of interest that should be locked in Interest script UTxO for the lender NFT holder to claim.

The rounding method of the computation is flooring. That means that the borrower is favoured.

Recommendation

We suggest using ceiling division instead. That way, the lender would be eligible for receiving interest also for the smallest chunk of time that hasn't entirely passed yet.

Resolution

We reported the issue but haven't heard back about it.

AADA-303: Liquidation checks too lenient

Severity:  Minor

Status: Resolved 

Description

In `Collateral.hs`, the checks for liquidation are too lenient and also allow minting of `liquidationNft` tokens. When oracle policy is not written carefully, it allows the owner of `dest` address to liquidate someone's collateral.

Recommendation

Disallow minting of `liquidationNft` tokens in `Collateral.hs`, so that mistakes in oracle policy do not propagate further.

AADA-304: Default liquidation policy relies on all 3 oracles working

Severity:  Minor

Status: Acknowledged

Description

In case of a private key being compromised or one oracle not working, the collateral can be unretrievable until the end of the loan period. By the time it might be worth almost nothing in case of a very volatile asset being used as the collateral.

There is an option to not use the default policy which makes this issue less severe.

Recommendation

For users: Do not use the default liquidation policy, as this can lock the collateral forever in a smart contract.

For AADA developers: provide a better default oracle liquidation policy design.

Resolution

According to the client, this will be improved in the next version when there'll be better oracle design.

AADA-305: Constant protocol staking key rewards are capped

Severity: ■ Minor

Status: Acknowledged

Description

The staking part of the contract address is now fixed by the protocol in the contract parameters. However, the rewards a particular stake pool can earn is capped. That means that once a certain amount accumulates in the protocol, the staking rewards will not grow. In fact, they can decrease.

Also, the staking pool that is chosen might have other funds which increase the total sum and thus potentially decrease the staking rewards. In that case a change of staking pool during the loan period would make sense.

Recommendation

Allow changing staking pool during the loan period.

Resolution

It has been decided to have a fixed pool in the first version of the application.

AADA-401: Dead Code

Severity:  Informational

Status: Resolved 

Description

There is some unused dead code.

In Utils.hs, there are functions valueIn and valueToSc that are not used anywhere.

In RequestDatum, fields lenderNftTn and lendDate are relevant and known just in CollateralDatum, not in RequestDatum. Having them in RequestDatum requires the borrower to supply invalid data that is either way not used.

Recommendation

Remove the dead code.

AADA-402: Inverted percentage calculation

Severity: ■ Informational

Status: Resolved ✓

Description

Inside function `validateInterestAmnt`, the interest calculation is wrong, using $interestamt \cdot 100 / interestPercentage$. And in `interestPercentage` function, `loanHeld` should be divided by `repayInterval`.

It works together by luck, as these mistakes cancel each other: $\frac{1}{\frac{1}{x}} = x$.

Recommendation

Fix the calculation or simplify the calculation into a one-liner that doesn't require percentages and unnecessary rounding (see issue ...).

AADA-403: Possible minting of oracle tokens with different token name

Severity:  Informational

Status: Resolved 

Description

There is no restriction about minting oracle tokens with different than hard-coded tn token name. It just needs to be minted alongside the token with the correct token name.

Recommendation

Enforce that only a specific constant token name oracle tokens can be minted.

AADA-404: Naming inconsistencies and improvements

Severity: ■ Informational

Status: Resolved ✓

Description

There are multiple inconsistencies in the naming. Namely:

- Across the base, the term NFT is used for any token. Not all NFTs in the code are actually non-fungible tokens.
- Camel case is not used in variables and function names consistently.
- The two partial validators in `Collateral.hs` are called `validateBorrower` and `validateLender`. We suggest naming them rather by the transaction type they validate, so `Return` and `Liquidation` respectively.

Recommendation

We suggest fixing the above points.

AADA-405: Duplicated logic

Severity: ■ Informational

Status: Pending

Description

There are multiple places where logic is unnecessarily duplicated. Namely:

- Scripts Liquidation.hs and Interest.hs are the same, naming aside
- mintFlattened utility function is re-implemented in AadaNft.hs

Recommendation

For a smaller codebase and less room for bugs, we recommend to de-duplicate the logic and reuse existing constructions.

That means refactoring Liquidation.hs and Interest.hs into a common Claim.hs script, reusing existing utility functions where possible.






Resolution

We reported the issue but haven't heard back about it.

Appendix


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 in code or in documentation
Acknowledged	Acknowledged by the project to be fixed later or out of scope
Pending	Still waiting for a fix or a official response