



# AADA

Audit Report, v1

September 2, 2022

**Disclaimer:** This report is presented without warranties or guarantees on the security of the code. We have conducted an analysis to the best of our ability in a limited period of time, therefore the list of issues is not guaranteed to be exhaustive.

# 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 amount. The created request carries locked collateral to guarantee the loan repayment. The 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 elsewhere. Responsibilities of the respective party are transferred alongside the NFT ownership.

The 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. The full interest amount is paid after the full agreed upon 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, the 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 off on the liquidation.

Other important details:

- The 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 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 the commit `77ef49bf35c45d5a8a442533f4562ea016ea12cf` that have been fixed by the design update itself.

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

Our manual process focused on several types of attacks, including but not limited to:

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 2 September 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 issues are marked in the report accordingly.

## Files audited

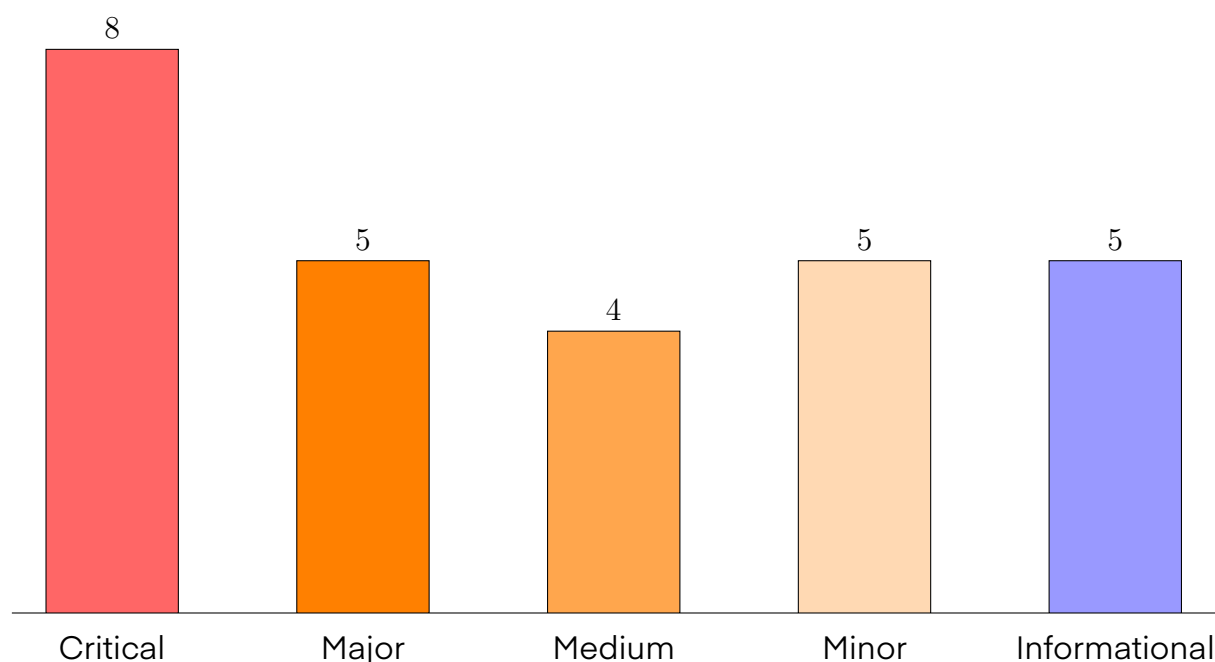
The files and their hashes reflect the final state at commit

19db8c9e03dff9a3f048d97c89b9b51720279ba after all the fixes have been implemented.

| SHA256 hash   | Filename            |
|---------------|---------------------|
| ee6b1...49129 | src/AadaNft.hs      |
| e3271...0d49e | src/Collateral.hs   |
| b4755...2032f | src/Common/Utils.hs |
| de723...01e93 | src/Interest.hs     |
| c6008...68df3 | src/Liquidation.hs  |
| 798fe...fb421 | src/OracleNft.hs    |
| 4c528...a4a30 | src/Request.hs      |

Please note that we did not audit the AADA staking files also present in the src/ folder at the final commit. These were manually added during our collaboration and are **out of scope of the audit**.

# Severity overview



## Findings

|          |  |   |              |
|----------|--|---|--------------|
| AADA-001 | Borrower/Lender tokens do not have to be NFTs                      | <span style="color: red;">■</span> Critical | Resolved ✓   |
| AADA-002 | Double Satisfaction – Collateral                                   | <span style="color: red;">■</span> Critical | Resolved ✓   |
| AADA-003 | Double Satisfaction – Loan   | <span style="color: red;">■</span> Critical | Resolved ✓   |
| AADA-004 | Resource DoS attack  | <span style="color: red;">■</span> Critical | Resolved ✓   |
| AADA-005 | Lender can take collateral when providing loan                     | <span style="color: red;">■</span> Critical | Resolved ✓   |
| AADA-006 | Lender can take the whole collateral in liquidation after deadline | <span style="color: red;">■</span> Critical | Acknowledged |
| AADA-007 | Inability to stop a compromised oracle node                        | <span style="color: red;">■</span> Critical | Acknowledged |
| AADA-008 | Oracle price consensus uncertainty                                 | <span style="color: red;">■</span> Critical | Acknowledged |
| AADA-101 | Borrower may repay loan without paying interest                    | <span style="color: orange;">■</span> Major | Resolved ✓   |

|          |  |               |                    |
|----------|--|---------------|--------------------|
| AADA-102 | Lender can collect interest immediately                          | Major         | Resolved ✓         |
| AADA-103 | Staking key can be freely modified                               | 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        | Partially resolved |
| AADA-204 | Lack of oracle decision transparency                             | Medium        | Acknowledged       |
| AADA-301 | Loss of Min-ADA  | Minor         | Acknowledged       |
| AADA-302 | Rounding of interest amount favors the Borrower NFT holder       | Minor         | Acknowledged       |
| AADA-303 | Liquidation token minted check too lenient                       | Minor         | Resolved ✓         |
| AADA-304 | Constant protocol staking key rewards are capped                 | Minor         | Acknowledged       |
| AADA-305 | Oracle policy not checking the destination of the tokens         | Minor         | Resolved ✓         |
| AADA-401 | Dead Code  | Informational | Partially resolved |
| AADA-402 | Inverted percentage calculation                                  | Informational | Resolved ✓         |
| AADA-403 | Possible minting of liquidation tokens with different token name | Informational | Resolved ✓         |
| AADA-404 | Naming inconsistencies and improvements                          | Informational | Partially resolved |
| AADA-405 | Duplicated logic   | Informational | Acknowledged       |

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

**Severity:** ■ Critical  
**Category:** Design Issue  
**Status:** Resolved ✓

## Description

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

As a result, 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, a malicious Borrower/Lender can use their second token to collect the collateral/interest/loan instead.

## Recommendation



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

## 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  
**Category:** Bug  
**Status:** Resolved 

## Description

The 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  
**Category:** Bug  
**Status:** Resolved ✓

## Description

The Lender can batch multiple loan requests in a single transaction. If multiple of those specify the same Borrower, The Lender may provide a loan only for the maximum of the wanted amounts (per loan currency). This effectively deprives the Borrower of money. Moreover, 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 another protocol's smart contracts. The Lender can include a loan request with any other smart contract that does not forbid it and that expects the Borrower to get paid. For example, the Borrower listing an item for sale on an NFT marketplace. The Lender could then pay only the highest expected amount - instead of the sum - and satisfy all of 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

**Category:** Bug

**Status:** Resolved ✓

## Description

The Borrower NFT holder can add additional useless tokens in the Return transaction, making the output UTxO too big. As a result, 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, the Lender NFT holder can add additional useless tokens making the Return transaction impossible to make for the 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 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  
**Category:** Bug  
**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 than or the same as 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 greater 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  
**Category:** Design Issue  
**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 leftover collateral is locked into the Liquidation script that can be claimed by the Borrower NFT holder later.

The liquidation done purely by the Lender NFT holder 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 leftover 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.

## Our comment

We advise Borrowers to make extra sure that the loan repayment transaction has sufficiently long time to pass before the deadline in order to minimize the risk of losing the collateral to the Lender NFT holder completely.

# AADA-007: Inability to stop a compromised oracle node

**Severity:** ■ Critical  
**Category:** Design Issue  
**Status:** Acknowledged

## Description

The provided default oracle policy is bound to a fixed set of nodes represented by public key hashes that all need to sign a liquidation transaction. If one of the nodes stops responding, ignores certain requests, loses the private key, etc, the node cannot be stopped and / or replaced. A new oracle policy defining different nodes could be used for the new loans, but there is no way to update the oracle policy for existing ones.

## Recommendation

We recommend having a more flexible node consensus mechanism. Instead of requiring all of the nodes to promptly agree with a liquidation, there could be a four out of five requirement or something similar.

Furthermore, we could replace public key hashes fixed in the contract with an oracle node token policy and distribute tokens instead. That way, there could be an even more flexible approach on who acts as an oracle node.

## Resolution

The client acknowledged the issue. There are plans to improve on the oracle design and the default oracle node policy is likely to be replaced.

# AADA-008: Oracle price consensus uncertainty

**Severity:** ■ Critical  
**Category:** Design Issue  
**Status:** Acknowledged

## Description

The ability for even fair oracle nodes to agree on a liquidation is uncertain. Oracle nodes are chosen to be decentralized exchanges at the moment. It's unclear how the nodes should determine and agree on an exchange rate that affects the liquidation. Assets can be highly volatile and with not that much liquidity, so the price could be easily manipulated. In the scenario of price moving on some exchanges, but not on all of them, there can be a lack of agreement on a liquidation with specific parameters.

## Recommendation

We recommend clearly defining how the oracle should decide on the exchange rate affecting liquidation, what price feeds it should watch and explore countermeasures for potential attacks.

Moreover, we recommend using only stable assets for collateral to minimize the risk.

## Resolution

The client acknowledged the issue. There are plans to improve on the oracle design later on.

# AADA-101: Borrower may repay loan without paying interest

**Severity:**  Major  
**Category:** Logical Issue  
**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 `interestPayDate` timestamp may be arbitrarily chosen and thus it may be set in such a way that the partial interest computation would result in zero interest. As a result, the Borrower NFT holder can repay the loan without paying interest. The issue lies in insufficient validation that checks that `interestPayDate` is a sufficiently accurate 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 NFT holder will have to pay enough interest.



# AADA-102: Lender can collect interest immediately

**Severity:**  Major  
**Category:** Logical Issue  
**Status:** Resolved 

## Description

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

The issue lies in insufficient time validation that checks that `lendDate` is a sufficiently accurate time approximation for the use case.

## Recommendation

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

# AADA-103: Staking key can be freely modified

**Severity:** ■ Major  
**Category:** Design Issue  
**Status:** Resolved ✓

## Description

There are multiple places where potentially big amounts of money are locked in the protocol contracts. It's important to define and control the party that is able to earn the staking rewards in case the collateral/loan/interest is in ADA.

Looking at the protocol, for example, we can see that the collateral is first locked into the Request contract by the Borrower, but it's consequently forwarded into the Collateral contract by the Lender. The current implementation does not check that the Lender does not modify the staking key for his own benefit. He can therefore earn staking rewards during the loan duration instead of the Borrower NFT holder or even the Lender NFT holder (note that the Lender and the Lender NFT holder are potentially different entities).

## Recommendation

As we could not find info about the above business requirements, we suggest defining them and enforcing them at the on-chain level.

## Resolution

The business requirement was defined to have a fixed protocol staking key as part of the contract parameters. Staking rewards should therefore go to the protocol. The stake pool is chosen by the AADA project. The requirement was enforced by the validators.

# AADA-104: Burning of oracle tokens can liquidate

**Severity:** ■ Major  
**Category:** Logical Issue  
**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.

Additionally, burning the default provided oracle token does not check the oracle node signatures. The default oracle policy 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 does, the check would drain `min-ADA` amount for every liquidation. If it's a person, they can consequently liquidate any collateral, making this critical from a 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.

## Resolution

The burning of liquidation tokens was forbidden in liquidation transactions. The simplification of the default oracle policy was not addressed.

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

**Severity:**  Major  
**Category:** Design Issue  
**Status:** Acknowledged

## Description

The validator does not enforce the structure of the liquidation transaction. This includes both types of liquidations – the liquidation with an oracle and the liquidation without an oracle.

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

## Recommendation

Enforce the proper structure of the liquidation transaction in the validator instead of relying on an 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

**Category:** Design Issue

**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 be unable 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 on time. He can then make a transaction liquidating the whole collateral and can, with some non-negligible probability, make it be applied first. The Borrower NFT holder 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.

In our opinion, one way to do this could be restricting the Lender NFT holder in taking the whole collateral upon liquidation. Instead, he may be able to take only a fair portion of it. The portion 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  
**Category:** Logical Issue  
**Status:** Resolved ✓

## Description

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

Computing the partial interest first computes the percentage of the elapsed time 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 computing the partial interest amount directly, with divisions being the last steps in the equation.

# AADA-203: Incomplete documentation

**Severity:** ■ Medium

**Category:** Documentation

**Status:** Partially resolved

## Description

There are places where documentation is incomplete or outdated. Some very important business requirements are not documented at all. Currently, the user needs to read the code to figure out the following:

- Which entity gets staking rewards for the locked assets
- That interest does not have to be present until the loan is repaid by the Borrower. Documentation in Gitbook is wrong on this.

Moreover, there is no public code for off-chain liquidation and documentation does not explain the exact rules for liquidation. For instance, datum parameters `collateralFactor` and `liquidationCommission` have unclear meanings as a result.

There are more examples. We are listing three that we find the most significant.

## Recommendation

First and foremost, we recommend documenting all business requirements, so that users know exactly what to expect from the application.

Furthermore, we suggest documenting how those business requirements translate to code with the aim of providing interested users with sufficient know-how to further check the implementation.

## Resolution

The system design figure in Gitbook is updated. Other than that, AADA team acknowledged that documentation is missing and promised to improve on it in the future.

# AADA-204: Lack of oracle decision transparency

**Severity:** ■ Medium

**Category:** Design Issue

**Status:** Acknowledged

## Description

The provided default oracle policy requires the signature of 3 oracle nodes in order to liquidate collateral. However, the decision of whether to liquidate or not is binary. The transaction is either signed or not. Users of the protocol and the protocol itself put their trust into the nodes and would like to verify that they're not misbehaving. Currently, this is hard to do as there are many factors why a node could decide to sign or not to sign a transaction.

## Recommendation

We suggest making the oracle node role 100% specified with clear guidelines on what is supposed to be liquidated and what's not. Put as much logic about the transaction structure into the validator (or the oracle minting policy) as possible.

Ideally, make the oracle node responsibilities simpler. Theoretically, it could just agree or disagree with a certain exchange rate present in the transaction in question as a redeemer. That way, anyone could use a blockchain explorer, find the transaction, see the redeemer and check that the exchange rate is correct. Increased transparency leads to increased security when the security depends on the oracle nodes risking their reputation only.

## Resolution

The client acknowledged the issue. There are plans to improve on the oracle design later on.



# AADA-301: Loss of Min-ADA

**Severity:** ■ Minor  
**Category:** Design Issue  
**Status:** Acknowledged

## Description

Min-ADA from the Borrower that is present in the Request UTxO can be ultimately taken away by the the Lender NFT holder in case all the tokens (Collateral, Interest, Loan) are not ADA and the Borrower needs to add min-ADA value to the Request UTxO. It also needs to be present in all the other script UTxOs and can ultimately be claimed by the Lender NFT holder upon a successful Claim transaction. This can be an inconsistency between the expectations of the Borrower and reality.

However, we are talking about approximately 2 ADA, making it a minor severity.

## Recommendation

Mitigation is again more of a business decision (that is not documented at the moment). 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

AADA team acknowledged the issue, but does not consider it a priority.

# AADA-302: Rounding of interest amount favors the Borrower NFT holder

**Severity:** ■ Minor  
**Category:** Design Issue  
**Status:** Acknowledged

## Description

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

The rounding method of the computation is flooring. That means that the Borrower NFT holder is favored.

## Recommendation

We suggest using ceiling division instead. That way, the Lender NFT holder would be eligible for receiving interest also for the smallest chunk of time that has not entirely passed just yet.

## Resolution

AADA team acknowledged the issue and does not plan to fix it at the moment as it is a minor one.

# AADA-303: Liquidation token minted check too lenient

**Severity:** ■ Minor  
**Category:** Logical Issue  
**Status:** Resolved ✓

## Description

In a transaction liquidating collateral before deadline, a liquidation token needs to be minted. In `Collateral.hs`, this check is too lenient. It checks that any token of the liquidation token currency symbol is minted. However, it does not check that it has the proper token name. In combination with other findings in this report and a lenient default oracle policy, a transaction that burns the proper token name liquidation token and mints a liquidation token with a different random token name passes all validations and liquidates any collateral.

If the oracle policy used is not written carefully (which happens to be the case with the default provided one), it allows the holder of liquidation tokens to liquidate freely.

## Recommendation

We suggest requiring that a specific liquidation token is minted to allow liquidation. By liquidation token, we mean a combination of currency symbol and a fixed token name.

Even though this may be indirectly fixed by addressing other related issues in the report, we recommend addressing this on multiple fronts so that mistakes in oracle policy do not propagate further.

## Resolution

The team fixed this issue by addressing other findings in this report. This issue is therefore considered resolved. However, requiring a fixed token name liquidation token is not enforced.

# AADA-304: Constant protocol staking key rewards are capped

**Severity:**  Minor  
**Category:** Design Issue  
**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 are capped. That means that once a certain amount accumulates in the protocol, the staking rewards will not grow. In fact, they can decrease.

## Recommendation


Using a fixed staking part of the address for all loans is discouraged because of the above reason. We suggest thinking of a way to rotate a number of different protocol staking keys.

One way to address this could be to create the requests every time with a randomly chosen one of them. The key should also not be part of the contract parameters but rather a part of the datum to not have a different contract validator hash per staking key.

## Resolution

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

# AADA-305: Oracle policy not checking the destination of the tokens

Severity:  Minor

Category: Bug

Status: Resolved 

## Description

The provided default oracle minting policy checks that the tokens are minted into `dest` address that is part of the policy parameters. However, an invariant that the oracle tokens are located only there does not necessarily hold.

It is enough that one oracle token goes there. It does not enforce anything about oracle tokens with different token names. Moreover, if the oracle token is also present in the transaction inputs, this check is insufficient as the oracle token may be minted to a different address. It is enough that one token would go to `dest` address in the transaction.

Note: The severity of this is tightly coupled with the ability to burn oracle tokens. Also, it depends on who owns `dest` address.

## Recommendation

We suggest addressing the other issue about not allowing the burning of oracle tokens to liquidate collateral. Then, the oracle policy may be simplified to check just signatures and not the destination of tokens.

## Resolution

The burning of oracle tokens is forbidden to liquidate collaterals. The default oracle policy was simplified, and the check was removed.

# AADA-401: Dead Code

**Severity:**  Informational

**Category:** Code Style

**Status:** Partially 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 only 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.

## Resolution

First point was addressed. Unused fields from `RequestDatum` were acknowledged, and it was decided not to remove them in order not to complicate the off-chain backend logic.

# AADA-402: Inverted percentage calculation

Severity: ■ Informational

Category: Bug

Status: Resolved ✓

## Description

Inside the `validateInterestAmnt` function in `Collateral.hs`, the interest calculation is wrong.

It computes  $\frac{\text{interestamt} \cdot 100}{\text{interestPercentage}}$ .

In addition to that, in `interestPercentage` function, `loanHeld` should be divided by `repayInterval` and not the other way around.

It works together by luck, as these mistakes cancel each other:

$$\frac{1}{\frac{1}{x}} = x$$

## Recommendation

We recommend fixing the calculation or simplifying the calculation into a one-liner that does not require percentages and unnecessary rounding.

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

**Severity:** ■ Informational

**Category:** Bug

**Status:** Resolved ✓

## Description

Even though the token name is supplied in the default oracle policy as a parameter, there is no restriction about minting oracle tokens with different than the hard-coded `tx` token name. It just needs to be minted alongside the token with the correct token name in a single transaction.

## Recommendation

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



# AADA-404: Naming inconsistencies and improvements

**Severity:** ■ Informational

**Category:** Code style

**Status:** Partially 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, `Return` and `Liquidation`, respectively.

## Recommendation

We suggest fixing the above points.

## Resolution

`OracleNft.hs` still mentions NFTs even though the liquidation tokens are fungible. Same with `liquidateNft` mentioned in `CollateralDatum`. Other than that, the points seem to be addressed.

# AADA-405: Duplicated logic

**Severity:** ■ Informational

**Category:** Code style

**Status:** Acknowledged

## 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` and `OracleNft.hs`

## Recommendation

For a smaller codebase and less room for bugs, we recommend de-duplicating the logic and reusing existing constructions.

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






## Resolution

AADA team prefers to keep the `Liquidation.hs` and `Interest.hs` scripts separate. We have not heard about the other point.

# 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.   |
| 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.               |



**Contact us:**

[audit@vacuumlabs.com](mailto:audit@vacuumlabs.com)