vacuumlabs

DRAFT

AADA

Audit Report, v1.1

March 21, 2023

Contents

Revision table

Report version	Report name	Date	Report URL
1.1	Aiken rewrite	2023-03-21	Full report link
1.0-debt-request	Debt request addition	2022-11-25	Full report link
1.0	Main audit	2022-09-02	Full report link

1 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.

The functionality reflects the state of the previous audit reports (including the presence of the acknowledged issues). The significant changes in the protocol in this version include:

- Changed the scripts from v1 version to v2.
- The minimum interest change is included and is in scope of this audit. A borrower needs to pay at least 20% of the total interest amount to the lender NFT holder, even if he repays the loan immediately.

Methodology

We conducted a deep manual audit of the app rewrite to Aiken. We started the audit at commit `99e594b92739f577179a1a20da57d95a0ef2dc26`.

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 8 March 2023 to 21 March 2023. We interacted on Discord and gave feedback in GitHub pull requests. The team fixed all but one issue that was acknowledged. The acknowledged issue is marked in the report accordingly.

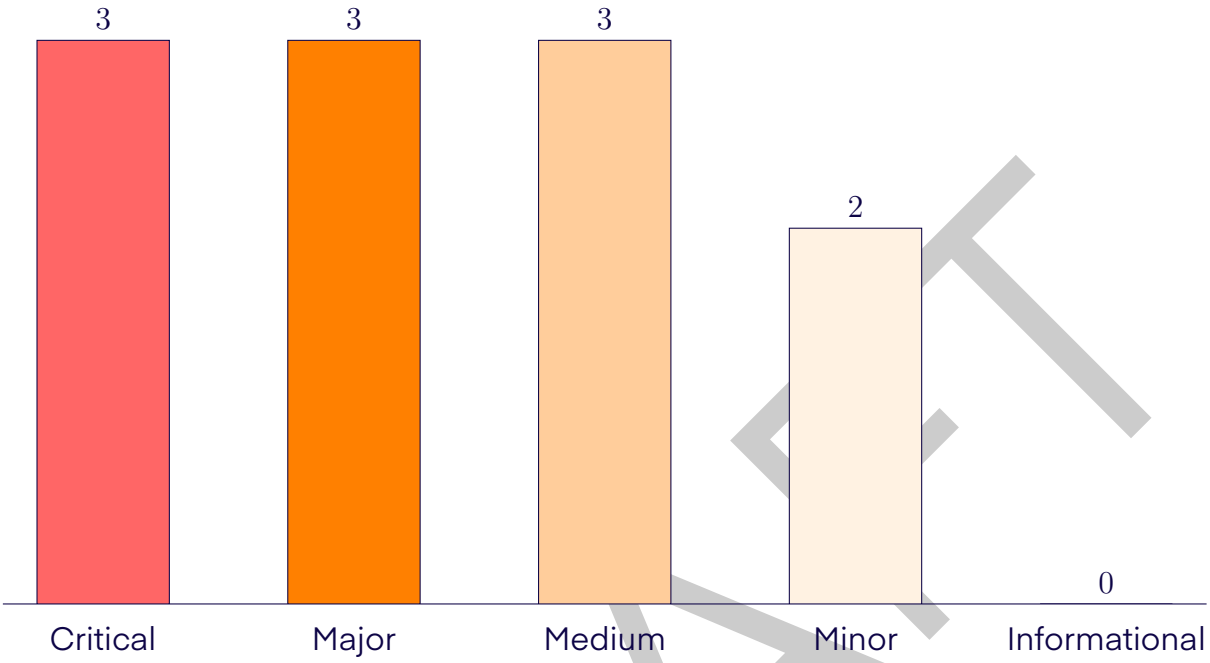
Files audited

The files and their hashes reflect the final state at commit `b3d7ccbe7967a511c76aa56e22fd82acb5377b17` after all the fixes have been implemented.

SHA256 hash	Filename
7abd2...62a73	lib/types.ak
317db...b0077	lib/utils.ak
79d78...1eea4	validators/aada_nft.ak
d4048...23367	validators/collateral.ak
62dcf...9f391	validators/interest.ak
25c9b...ad4dd	validators/liquidation.ak
f5843...36bbf	validators/request.ak
01daa...ff0e8	validators/request_debt.ak

Please note that we did not audit the files **not** listed above that are part of the commit hash. We also did not assess the security of Aiken language itself. The assessment builds on the assumption that Aiken is secure and delivers what it promises.

2 Severity overview



Findings

ID	TITLE	SEVERITY	STATUS
AADAIK-001	Attacker can spoof AADA NFTs and impersonate their rightful holders	CRITICAL	RESOLVED
AADAIK-002	Lender can liquidate and take the whole collateral as soon as he lends	CRITICAL	RESOLVED
AADAIK-003	Borrower is not able to repay the loan and loses the collateral	CRITICAL	RESOLVED
AADAIK-101	Borrower can pay only min interest for any loan	MAJOR	RESOLVED
AADAIK-102	Double satisfaction among different scripts	MAJOR	RESOLVED
AADAIK-103	Lender can collect interest immediately	MAJOR	RESOLVED

Continued on next page

ID	TITLE	SEVERITY	STATUS
AADAIK-201	Expiration setting of borrow request is not enforced	MEDIUM	RESOLVED
AADAIK-202	Interest calculation is imprecise	MEDIUM	RESOLVED
AADAIK-203	ADA locked by the protocol can not be staked	MEDIUM	RESOLVED
AADAIK-301	The collateral amount is set by the lender for the borrower	MINOR	ACKNOWLEDGED
AADAIK-302	Other code suggestions	MINOR	RESOLVED

AADAIK-001 Attacker can spoof AADA NFTs and impersonate their rightful holders

Category	Vulnerable commit	Severity	Status
Logical Issue	99e594b927	CRITICAL	RESOLVED

Description

The core of the problem is the `validate_token_mint` function. For a specific currency symbol, token name and amount, it validates that such a tuple is minted in the transaction. It does not check, though, that **only** such a tuple is minted. The Plutus code always checked the exclusivity of the tuple by pattern matching the flattened transaction mint value to a single such tuple.

An attacker can:

- Create a transaction minting a borrower NFT normally.
- In the same transaction, the attacker mints also a duplicate lender NFT for an existing loan.
- The validation is successful as it does not check the exclusivity of the minted tuple and does not check any by-products.
- The attacker waits for the borrower to repay the loan with interest into a smart contract.
- The attacker claims the value using the maliciously minted duplicate lender NFT instead of the true lender.

The above is just a sample working attack using this bug. Another could be minting a duplicate borrower NFT, repaying the loan prematurely instead of the true borrower and claiming the collateral. The assumption is that the collateral is bigger in value compared to the loan with interest.

Note that there are multiple other places where the problematic function is used. We did not investigate all the potential impacts.

Recommendation

We suggest rewriting the function to use a strict list pattern matching for a single minted tuple, similar to the Plutus implementation.

Resolution

Fixed in the pull request number 6.

AADAIK-002 Lender can liquidate and take the whole collateral as soon as he lends

Category	Vulnerable commit	Severity	Status
Bug	99e594b927	CRITICAL	RESOLVED

Description

The function `validate_liquidate` checks if the loan deadline is after the lower bound of the transaction validity range. As a result, it can be set to any timestamp in the past, so an attacker can always make the `deadline_passed` equal to `true`.

This gives the lender the possibility to liquidate the loan whenever he chooses so, even just right after the loan is given. He can take the whole collateral which is supposed to be bigger in value compared to the loan.

Recommendation

The validator should check that the liquidation transaction happens after the deadline has passed. Therefore, the check should pass only if the lower bound of the validity range is higher than the loan deadline.

Resolution

Fixed in the pull request number 6.

AADAIK-003 Borrower is not able to repay the loan and loses the collateral

Category	Vulnerable commit	Severity	Status
Bug	99e594b927	CRITICAL	RESOLVED

Description

The functions `get_expected_interest` and `validate_repay` use the asset name of a token instead of the policy ID in the `from_asset` function call. Because such a policy ID does not have to exist, the validator may not be able to pass and the loans may not be able to be repaid.

Recommendation

The functions should call the `from_asset` function correctly by supplying both the `policy_id` and the `asset_name`.

Resolution

Fixed in the pull request number 6.

AADAIK-101 Borrower can pay only min interest for any loan

Category	Vulnerable commit	Severity	Status
Bug	99e594b927	MAJOR	RESOLVED

Description

Because the `current_repay_time` uses the lower bound of the transaction validity range, a user can pass any time in the past as the `current_repay_time`. The user can for example set it to be equal to the `datum.lend_time` and since the interest is computed from this time, they will only need to pay the minimal interest.

Recommendation

The function should use the upper bound of the validity range to determine the `current_repay_time` so that the user can not set it into the past and bypass paying interest.

Resolution

Fixed in the pull request number 6.

AADAIK-102 Double satisfaction among different scripts

Category	Vulnerable commit	Severity	Status
Bug	99e594b927	MAJOR	RESOLVED

Description

The `Request` and `DebtRequest` validators prevent a double satisfaction attack in the `get_own_validator_inputs` function. This prevents double satisfaction just among multiple script inputs from the same protocol, though.

An attacker can mix a loan request of a Borrower (or a debt request of a Lender) 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 same party 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.

This issue is similar to AADA-003 from the version 1.0 of the AADA report. The main difference is that in AADA-003, the validator did not check the issue at all. Here, it checks that two of the same smart contracts are not among the inputs. This check is incomplete. Because the exploit possibilities are more limited, though, we downgraded the severity from critical to major.

Recommendation

We suggest making sure the loan request and debt request are always the only smart contracts in the transaction inputs.

Resolution

Fixed in the pull request number 6.

AADAIK-103 Lender can collect interest immediately

Category	Vulnerable commit	Severity	Status
Bug	99e594b927	MAJOR	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 they 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 `lend_time` is a sufficiently accurate time approximation for the use case.

Recommendation

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

Resolution

Fixed in the pull request number 6.

AADAIK-201 Expiration setting of borrow request is not enforced

Category	Vulnerable commit	Severity	Status
Bug	99e594b927	MEDIUM	RESOLVED

Description

The `validate_expiration` function in `utils.ak` uses the lower bound of the validity range to check for the request expiration. The lower bound can be set to any time in the past, meaning that the expiration check can be ignored by an attacker. The previous Plutus version of the smart contract used the upper bound, too.

The borrower might use the expiration deadline to e.g. protect the collateral from falling in value too much and thus getting liquidated. However, the smart contract may not adhere to the expiration setting.

Recommendation

Use the upper bound as in the Plutus version.

Resolution

Fixed in the pull request number 7.

AADAIK-202 Interest calculation is imprecise

Category	Vulnerable commit	Severity	Status
Logical Issue	99e594b927	MEDIUM	RESOLVED

Description

The partial interest computation goes through a division step, losing precision on the way. Plutus and Aiken both use `Integer` types; thus any division is flooring the (intermediary) result.

Then the number is multiplied by the actual interest amount. As a result, the partial interest is unnecessarily rounded, making the computation imprecise with the Lender NFT holder losing on this imprecision. The bigger the interest amount, the bigger the loss.

This is a similar issue to AADA-202 from the version 1.0 of the AADA report.

Recommendation

We suggest abandoning the intermediary rounding altogether and computing the partial interest amount directly with divisions being the last step in the equation.

Resolution

Fixed in the pull request number 6.

AADAIK-203 ADA locked by the protocol can not be staked

Category	Vulnerable commit	Severity	Status
Design Issue	99e594b927	MEDIUM	RESOLVED

Description

The contract addresses are constructed using the `utils.scripthash_address` utility function which inserts `None` as the staking credential to construct the script address. The following validation checks that the script output is at that address.

Additionally, even a loan request can not include the staking credential as the validator expects a single script input with no staking credential.

The previous Plutus implementation did not enforce anything in regards to the staking credential on the inputs and expected a specific contract address to be used as an output. That address could have contained a valid staking credential. The current Aiken translation does not enable this possibility.

Recommendation

We suggest keeping the Plutus logic in regards to the staking credentials.

Resolution

Fixed in the pull request number 6. The scripts now take the staking credential as a parameter. Thus the handling is the same as in the Plutus version.

However, there is one difference in the handling. The staking credential on the request and debt request UTxOs is now enforced. If somebody creates a UTxO and inserts another staking credential there, he was able to proceed with the old Plutus version. He is not able to proceed in this version but needs to cancel the request instead.

AADAIK-301 The collateral amount is set by the lender for the borrower

Category	Vulnerable commit	Severity	Status
Design Issue	99e594b927	MINOR	ACKNOWLEDGED

Description

Currently, the lender sets the collateral amount for the borrower at the moment of a debt request creation. The collateral amount is specified in the datum and needs to be copied over. The borrower can provide more collateral than needed, but it does not affect the amount written into the datum. This can lead to a potential liquidation happening much sooner compared to the case if the full value of the actually deposited collateral was taken into account. Provided the borrower actually deposited more collateral.

Recommendation

Let the borrower decide on the collateral amount. The lender can specify the minimum collateral amount and currency. This requires updating the `CollateralDatum` with the amount of the actually deposited collateral.

Resolution

The team acknowledges the problem as previously no user has used this feature, so it will not be reimplemented in this version.

AADAIK-302 Other code suggestions

Category	Vulnerable commit	Severity	Status
Code Style	99e594b927	MINOR	RESOLVED

Description

A few other code suggestions:

- **Naming:**
  - The `CurrencySymbol` type name may confuse readers as `CurrencySymbol` is generally used in a different context. We suggest using the `AssetClass` name instead.
  - In the `validate_repay` function, the `burned_lender_nft` variable should be called `burned_borrower_nft` instead.
  - In the `paid_borrower` function in the `request.ak` file, `tx_output` argument contains the list of all the transaction outputs. We suggest renaming it to reflect the plural in the name, e.g. to `tx_outputs`.
  - The variable `string_to_hash` in the `aada_nft.ak` file does not reflect its contents. We suggest renaming to e.g. `consumed_utxo_identifier`.
- **Dead code:** The functions `get_output`, `get_input`, `validate_expiration` (the one defined in the `collateral.ak` file) are not used. We suggest removing them.
- **Other:**
  - Unused redeemer parameters could be prefixed with `_redeemer`. This would remove Aiken warnings about unused variables.
  - Contract parameters such as Aada NFT currency symbol and contract addresses could be defined in a single place to minimize the chance of an error that could occur by repeating the same constants across the validators.

Recommendation

Recommendations are part of the issues description.

Resolution

All suggestions are fixed in the pull request number 6.

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 WEB3OU (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 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.

C 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.

v1.1: Aiken rewrite

Revision date: 2023-03-21
Final commit: `b3d7ccbe7967a511c76aa56e22fd82acb5377b17`

We conducted an audit of the on-chain code rewrite from Plutus to Aiken along with minor modifications to the protocol logic. To see the files audited, see Executive Summary.

Full report for this revision can be found at url.

v1.0-debt-request: Debt request addition

Revision date: 2022-11-25
Final commit: `2b2a170d3bac83f0784d7110131cd6e52c68e390`

We audited the addition of another request type (a *Lender* offering a loan).

SHA256 hash	Filename
`a714c...f839b`	`src/DebtRequest.hs`

Please note that any changes to other files have not been audited.

Full report for this revision can be found at url.

v1.0: Main audit

Revision date: 2022-09-02
Final commit: `19db8c9e03dffd9a3f048d97c89b9b51720279b`

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`

Full report for this revision can be found at url.

D About Us

Vacuumlabs has been building crypto projects since the day they became possible on the Cardano blockchain.

- Helped create the decentralized exchange on Cardano – WingRiders, currently the second largest exchange on Cardano (based on TVL).
- We are the group behind the popular AdaLite wallet. It was later improved into a multichain wallet named NuFi which also integrates a decentralised exchange.
- We built the Cardano applications for 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 & transactions.

Our auditing team is chosen from the best.

- Ex-WingRiders, ex-NuFi developers
- Experience from Google, Spotify, traditional finance, trading and ethical hacking
- Medals and awards from programming competitions: ACM ICPC, TopCoder, International Olympiad in Informatics
- Passionate about Program correctness, Security, Game theory and Blockchain

vacuumlabs

DRAFT

Contact us:
audit@vacuumlabs.com