



AADA Debt Request

Audit Report, v1.0

November 25, 2022

Contents

Disclaimer	1
1 Severity overview	6
AADADR-301: Lender sets the collateral amount for the borrower	7
AADADR-401: Branch the validation logic depending on a re- deemer	8
AADADR-402: <code>borrowerGetsWhatHeWants</code> is not used	9
AADADR-403: Redundant function for checking transaction in- puts	10
Appendix	11

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 WEB 3 OU (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 hyper-linked 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.**

Executive summary

Note: THIS AUDIT ONLY CONCERNS THE DEBT REQUEST PART OF THE AADA PROJECT. READ THE [AADA v1 AUDIT](#) FOR MORE INFORMATION ABOUT THE REST OF THE PROJECT.

Project overview

The AADA project offers peer-to-peer decentralized lending. A person interested in lending assets (i.e. **Lender**) can create a debt request with information about the loan including its duration and interest. The created request carries the loan amount locked in the smart contract. The Lender gets a bond NFT (**Lender NFT**) in exchange for creating the request. It can later be used to cancel the request or to collect the repayment along with interest.

Someone else can fulfill the request (i.e. **Borrower**) by locking the requested collateral amount in the smart contract and taking the lent assets from the smart contract. They get back a bond NFT (**Borrower NFT**) proving that they locked the collateral.

Both bond NFTs – 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.

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, 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 **Lender** in the debt request. In the default policy provided, three **oracle nodes** need to sign the liquidation off.

Other important details:

- We emphasize that the liquidation policy is chosen by the Lender in the debt request. It is up to the Borrower to check that he trusts it before depositing collateral.

- 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 the audit was helping the client to design the time handling in the debt request, so that the start of the loan is not set too much in the favor of the borrower. In the second phase, we conducted a deeper manual audit of the code, reported findings to the team and re-reviewed the fixes.

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 25 October 2022 to 25 November 2022. We interacted on Discord and gave feedback in GitHub pull requests. All reported issues have been resolved.

Files audited

The file and its hash reflect the final state at commit

2b2a170d3bac83f0784d7110131cd6e52c68e390 after all the fixes have been implemented.

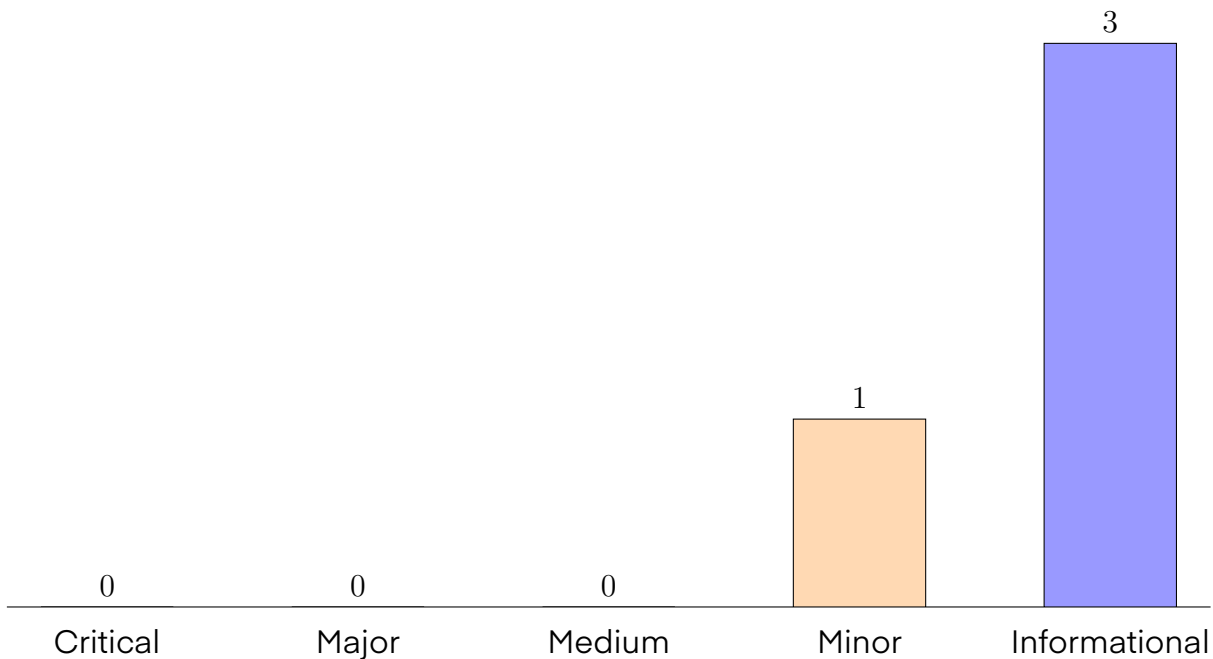
SHA256 hash	Filename
a714c...f839b	src/DebtRequest.hs

Previously audited files

Please note that we have only audited the file `src/DebtRequest.hs` in this audit. We assume that it will be used together with the smart contracts that have been part of the [previous audit](#). The file hashes of the relevant files are listed here for completeness.

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

1 Severity overview



Findings

AADADR-301	Lender sets the collateral amount for the borrower	Minor	Resolved ✓
AADADR-401	Branch the validation logic depending on a redeemer	Informational	Resolved ✓
AADADR-402	<code>borrowerGetsWhatHeWants</code> is not used	Informational	Resolved ✓
AADADR-403	Redundant function for checking transaction inputs	Informational	Resolved ✓

AADADR-301: Lender sets the collateral amount for the borrower

Severity:  Minor
Category: Design Issue
Status: Resolved 

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

We suggest letting the borrower decide on the collateral amount with the lender just being able to specify the minimum collateral amount and currency. This requires updating the `CollateralDatum` with the amount of the actually deposited collateral.

Resolution

Fixed in the pull request number 59 according to our recommendation.

AADADR-401: Branch the validation logic depending on a redeemer

Severity: ■ Informational

Category: Code Style

Status: Resolved ✓

Description

The code uses a validation of the form $(a_1 \wedge a_2 \wedge \dots \wedge a_k) \vee (b_1 \wedge \dots \wedge b_l)$, where the first part $a_1 \wedge a_2 \wedge \dots \wedge a_k$ corresponds to a fulfilled request and $b_1 \wedge \dots \wedge b_l$ corresponds to a canceled request.

However, this type of code is often harder to read and could be also more expensive to run.

Recommendation

Using a redeemer for branching of the validation logic has the following advantages.

- It can lower transaction costs as there are fewer conditions evaluated per redeemer.
- It makes the code easier to read (and write), making it easier to change in the future and less prone to errors.

Resolution

Fixed in the pull request number 60 according to our recommendation.

AADADR-402: `borrowerGetsWhatHeWants` is not used

Severity:  Informational

Category: Code Style

Status: Resolved 

Description

The function `borrowerGetsWhatHeWants` is not used.

Recommendation

We suggest removing `borrowerGetsWhatHeWants`.

Resolution

Fixed in the pull request number 60 according to our recommendation.

AADADR-403: Redundant function for checking transaction inputs

Severity:  Informational

Category: Code Style

Status: Resolved 

Description

The function `txHasOneDebtRequestInputOnly` checks only a subset of what `txHasOneScInputOnly` already checks. There is no harm in leaving it like that, but it is not necessary.

Recommendation

We suggest removing the `txHasOneDebtRequestInputOnly` check.






Resolution

Fixed in the pull request number 60 according to our recommendation.

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