FYEO

Security Assessment for 3A Borrowing Protocol

3Adao.org

August 2023
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Public

Table of Contents

List of Figures

List of Tables

EXECUTIVE SUMMARY

Overview

3Adao.org engaged FYEO Inc. to perform a Security Assessment for 3A Borrowing Protocol.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on July 31 - August 25, 2023, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-3A-ID-01 – Lack of Validation for _collateral address in contracts/Vault.sol
- FYEO-3A-ID-02 – Unchecked Return Values contracts/A3AStaking.sol
- FYEO-3A-ID-03 – Unrestricted Ownership Transfer contracts/LastResortLiquidation.sol
- FYEO-3A-ID-04 – Unrestricted Ownership Transfer contracts/LiquidationRouter.sol
- FYEO-3A-ID-05 – Unlimited Approval Risk A3AStaking.sol
- FYEO-3A-ID-06 – potential divide by zero in reward calculation StabilityPool.sol

Based on our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Assessment for 3A Borrowing Protocol. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/bonq-ch/3a-borrowing-protocol with the commit hash main.

Files included in the code review

```
3a-borrowing-protocol/
└── contracts/
    ├── bots/
    │   └── VaultOptimizerBot.sol
    ├── interfaces/
    │   ├── IAuctionManager.sol
    │   ├── IBONQStaking.sol
    │   ├── IExternalPriceFeed.sol
    │   ├── IFeeRecipient.sol
    │   ├── ILastResortLiquidation.sol
    │   ├── ILiquidationRouter.sol
    │   ├── IMintableToken.sol
    │   ├── IMintableTokenOwner.sol
    │   ├── IOwnable.sol
    │   ├── IPriceFeed.sol
    │   ├── IRouter.sol
    │   ├── IStabilityPool.sol
    │   ├── ITokenPriceFeed.sol
    │   ├── IVault.sol
    │   ├── IVaultDeployer.sol
    │   ├── IVaultFactory.sol
    │   ├── IVaultFactoryConfig.sol
    │   └── IWETH.sol
    ├── oracles/
    │   ├── ChainlinkPriceFeed.sol
    │   ├── ConvertedPriceFeed.sol
    │   └── MockConvertedPriceFeed.sol
    ├── utils/
    │   ├── BONQMath.sol
    │   ├── PoolAddress.sol
    │   ├── constants.sol
    │   └── linked-address-list.sol
    ├── A3A.sol
    ├── A3AStaking.sol
    ├── APToken.sol
    ├── AuctionManager.sol
    ├── LastResortLiquidation.sol
    ├── LiquidationRouter.sol
    ├── MintableToken.sol
    ├── MintableTokenOwner.sol
```

Files included in the code review
├── StabilityPool.sol ├── TokenToPriceFeed.sol ├── Vault.sol ├── VaultDeployer.sol ├── VaultFactory.sol ├── VaultFactoryConfig.sol └── VaultFactoryList.sol

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

During the Security Assessment for 3A Borrowing Protocol, we discovered:

- 4 findings with MEDIUM severity rating.
- 2 findings with LOW severity rating.

The following chart displays the findings by severity.

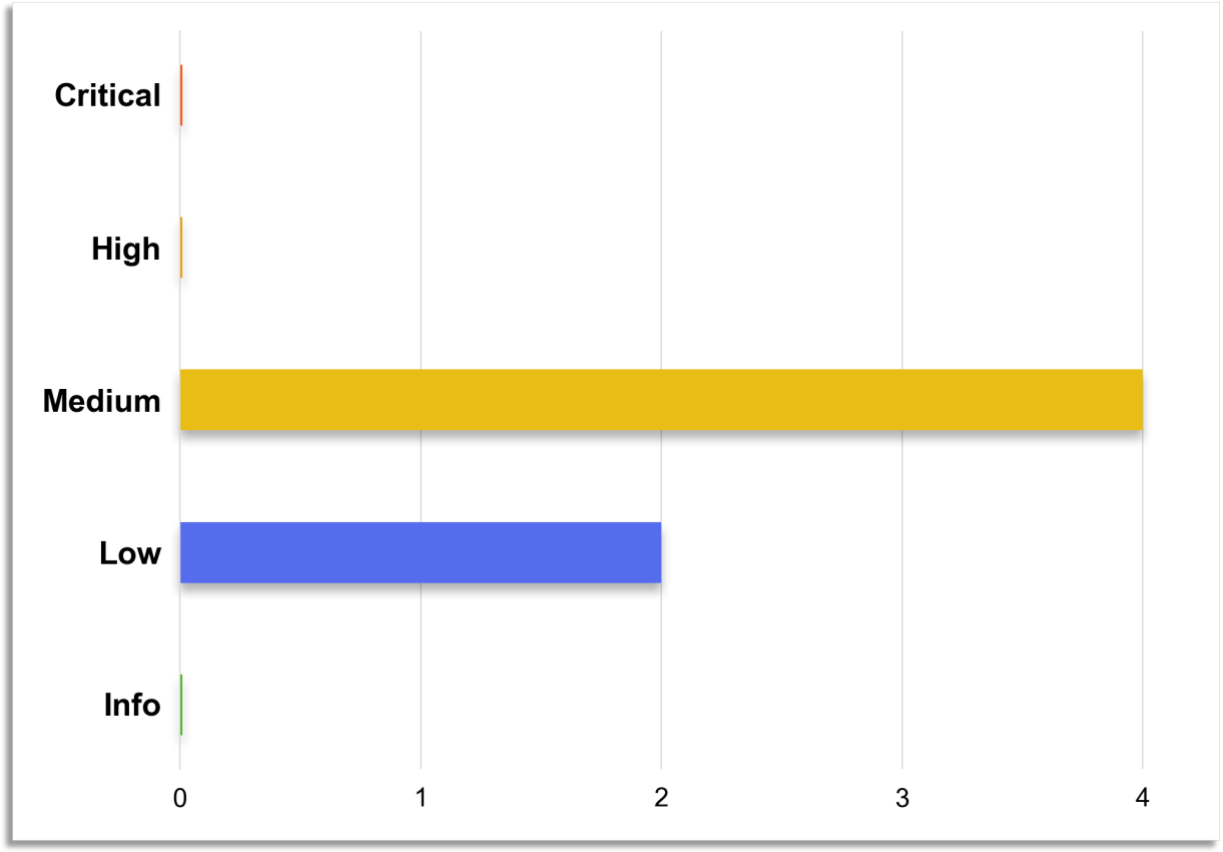


Figure 1: Findings by Severity

FINDINGS

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
FYEO-3A-ID-01	**Medium**	Lack of Validation for _collateral address in contracts/Vault.sol
FYEO-3A-ID-02	**Medium**	Unchecked Return Values contracts/A3AStaking.sol
FYEO-3A-ID-03	**Medium**	Unrestricted Ownership Transfer contracts/LastResortLiquidation.sol
FYEO-3A-ID-04	**Medium**	Unrestricted Ownership Transfer contracts/LiquidationRouter.sol
FYEO-3A-ID-05	**Low**	Unlimited Approval Risk A3AStaking.sol
FYEO-3A-ID-06	**Low**	potential divide by zero in reward calculation StabilityPool.sol

Table 2: Findings Overview

TECHNICAL ANALYSIS

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

CONCLUSION

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

TECHNICAL FINDINGS

GENERAL OBSERVATIONS

Background

3A is a non-custodial, decentralized, over-collateralized lending platform tailored for projects and protocols with tokens. The platform uniquely addresses four major challenges:

1. Enables borrowing against self-owned tokens at a zero-interest rate, reducing the financial strain.
2. Introduces a liquidity solution without mandating incentives or payments to the liquidity pool's opposite side.
3. Proposes a sustainable yield to community token holders, prioritizing safety and security.
4. Facilitates treasuries to adopt a risk-averse approach, promoting intelligent capital allocation.

Code Review Summary

The smart contracts associated with **3A** have been well crafted, reflecting high quality and adherence to best security practices:

- **Access Control**: Critical functions are protected using the Ownable or Factory modifier, ensuring that only the designated owner can execute them. A custom modifier, onlyAllowed, is also in place for specific functions, allowing access exclusively to addresses listed in the allowedSet.
- **Reentrancy Attacks**: The smart contracts are well protected against reentrancy attacks through the utilization of the ReentrancyGuard throughout the suite of contracts.
- **Safe Token Transfers**: Leveraging the SafeERC20 module from OpenZeppelin, the contracts ensure robust and secure token transfers, safeguarding against reentrancy and addressing potential inconsistencies inherent to some ERC20 tokens.

LACK OF VALIDATION FOR _COLLATERAL ADDRESS IN CONTRACTS/VAULT.SOL

Finding ID: FYEO-3A-ID-01
Severity: **Medium**
Status: **Remediated**

Description

The contract does not validate the `_collateral` address in the `borrowableWithDiff` function. This could lead to unexpected behavior if the function is called with an address that is not a valid ERC20 token.

Proof of Issue

File name: contracts/Vault.sol
Line number: 252

```
function borrowableWithDiff(
    address _collateral,
    uint256 _diffAmount,
    bool _isAdd,
    bool _useMlr
) public view returns (uint256 _maxBorrowable, uint256 _borrowable) {
    require(_collateral != address(0x0), "collateral-is-0");
    //...
}
```

Severity and Impact Summary

If exploited, it could lead to unexpected behavior and potential loss of user funds.

Recommendation

Validate the `_collateral` address before using it. Ensure it is a valid ERC20 token.

UNCHECKED RETURN VALUES CONTRACTS/A3ASTAKING.SOL

Finding ID: FYEO-3A-ID-02
Severity: **Medium**
Status: **Remediated**

Description

The contract does not check the return value of the `transferFrom` and `transfer` functions of the ERC20 token contract.

Proof of Issue

File name: A3AStaking.sol
Line number: 114, 142

```
a3aToken.transfer(msg.sender, A3AToWithdraw);

stableCoin.transferFrom(msg.sender, address(this), _amount);
```

Severity and Impact Summary

This could lead to unexpected behavior if these functions fail. Or if a stable coin in this context wished to act maliciously- If that particular implementation of ERC20 does return false to indicate a failure (instead of reverting), this contract would continue executing as though it succeeded. This could lead to the contracts internal state being untrue or out of state.

Recommendation

Always check the return value of `transferFrom` and `transfer` functions. It's a good practice to handle these potential failures appropriately to prevent any unexpected behavior. We recommend using the using SafeERC20 functions `safeTransfer` `safeTransferFrom` `safeApprove` instead of the standard functions.

UNRESTRICTED OWNERSHIP TRANSFER IN LASTRESORTLIQUIDATION

Finding ID: FYEO-3A-ID-03
Severity: **Medium**
Status: **Remediated**

Description

The contract `LastResortLiquidation` inherits from the `Ownable` contract provided by OpenZeppelin. However, it does not restrict who can become the owner of the contract.

Proof of Issue

File name: LastResortLiquidation.sol
Line number: 1

```
contract LastResortLiquidation is Ownable, ReentrancyGuard {
...
}
```

Severity and Impact Summary

This is a potential security risk as the owner has the power to perform sensitive operations such as adding or removing allowed addresses, setting the vault factory, withdrawing collateral and repaying bad debt.

Recommendation

Implement additional checks to ensure that the ownership of the contract cannot be transferred to an arbitrary address. This could include requiring the new owner to be an existing member of a whitelist, or requiring multiple signatures to approve the transfer.

UNRESTRICTED OWNERSHIP TRANSFER CONTRACTS/LIQUIDATIONROUTER.SOL

Finding ID: FYEO-3A-ID-04
Severity: **Medium**
Status: **Remediated**

Description

The contract is using the `Ownable` contract from OpenZeppelin which allows the contract owner to transfer ownership to any other address. However, there is no restriction or validation on the new owner's address.

Proof of Issue

File name: LiquidationRouter.sol
Line number: 1

```
contract LiquidationRouter is Ownable, ReentrancyGuard {
...
}
```

Severity and Impact Summary

This could lead to accidental loss of contract control if the owner's address is mistakenly set to an incorrect address.

Recommendation

Implement additional checks to prevent the owner from transferring ownership to the zero address or to the contract address itself. This could be done by overriding the `transferOwnership` function in the `Ownable` contract.

UNLIMITED APPROVAL RISK A3ASTAKING.SOL

Finding ID: FYEO-3A-ID-05
Severity: **Low**
Status: **Remediated**

Description

The function "redeemReward" sets an unlimited approval on the stableCoin token for the factory contract. This could potentially allow the factory contract to drain all the stableCoin tokens from this contract.

Proof of Issue

File name: A3AStaking.sol
Line number: 153

```
function redeemReward(uint256 _amount, address _vaultAddress) external {
    //...
    stableCoin.approve(address(factory), MAX_INT);
    //...
}
```

Severity and Impact Summary

Loss of all stableCoin tokens. For now the factory contract is trusted and audited so the risk is low.

Recommendation

Only approve the exact amount that is needed for the operation.

POTENTIAL DIVIDE BY ZERO IN REWARD CALCULATION STABILITYPOOL.SOL

Finding ID: FYEO-3A-ID-06
Severity: **Low**
Status: **Remediated**

Description

in _computeRewardsPerUnitStaked There is a potential for a division-by-zero error. The code divides by `_totalStableCoinDeposits` multiple times, but it never explicitly checks if `_totalStableCoinDeposits` is zero.

Even though the assert statement checks for `_debtToOffset <= _totalStableCoinDeposits`, it doesn't guarantee `_totalStableCoinDeposits` isn't zero.

Proof of Issue

File name: StabilityPool.sol
Line number: 636

```
} else {
            uint256 stableCoinLossNumerator = _debtToOffset * DECIMAL_PRECISION -
lastStableCoinLossErrorOffset;
            /*
             * Add 1 to make error in quotient positive. We want "slightly too much"
StableCoin loss,
             * which ensures the error in any given compoundedStableCoinDeposit favors
the Stability Pool.
             */
            stableCoinLossPerUnitStaked = stableCoinLossNumerator /
_totalStableCoinDeposits + 1;
            lastStableCoinLossErrorOffset = stableCoinLossPerUnitStaked *
_totalStableCoinDeposits - stableCoinLossNumerator;
        }

        collateralGainPerUnitStaked = collateralNumerator / _totalStableCoinDeposits;
        collateralToLastErrorOffset[_collateralTokenAddress] =
            collateralNumerator -
            collateralGainPerUnitStaked *
            _totalStableCoinDeposits;

        return (collateralGainPerUnitStaked, stableCoinLossPerUnitStaked);
```

Severity and Impact Summary

Could lead to unexpected behavior in the reward calculation.

Recommendation

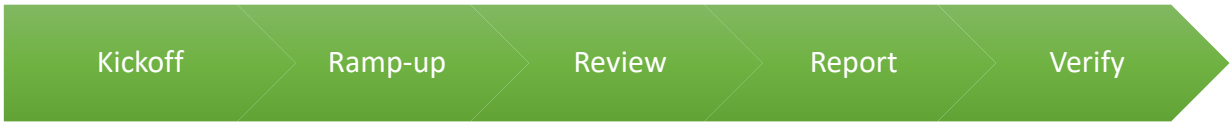
Check for non-zero before second division.

OUR PROCESS

METHODOLOGY

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2: Methodology Flow



KICKOFF

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

RAMP-UP

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

REPORTING

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

VERIFY

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

ADDITIONAL NOTE

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

THE CLASSIFICATION OF VULNERABILITIES

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations