FYEO

Security Assessment of the Rain Program

Rain.Fi

April 2023
Version 1.0

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

Security Level
Strictly Confidential

TABLE OF CONTENTS

LIST OF FIGURES

List of Tables

EXECUTIVE SUMMARY

Overview

Rain.Fi engaged FYEO Inc. to perform a Security Assessment of the Rain Program.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on February 22 - March 20, 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 were identified during the testing period and have since been remediated

- FYEO-RF-01 – Use-after-free due to a lifetime error in Vec::into_iter()
- FYEO-RF-02 – Casting Integer literal to 'u128' is unnecessary
- FYEO-RF-06 – Unnecessary equality check against true

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 of the Rain Program. 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/rain-foundation/rain-program/tree/merging with the commit hash 9f5f871c0f132e09bffc8d1241a9936d9fe8e9aa.

A re-review was carried out on commit hash: 59a55424be8efcc11f2dba7a625875f48cd21f10

Files included in the code review

```
rain-program/
├── programs/
│   └── rain/
│       ├── src/
│       │   ├── collection/
│       │   │   ├── create_collection.rs
│       │   │   ├── delete_collection.rs
│       │   │   ├── mod.rs
│       │   │   └── update_collection.rs
│       │   ├── loan/
│       │   │   ├── marketplace/
│       │   │   │   ├── auction_house.rs
│       │   │   │   ├── hadeswap.rs
│       │   │   │   ├── mod.rs
│       │   │   │   └── solanart.rs
│       │   │   ├── bot_liquidate.rs
│       │   │   ├── freeze.rs
│       │   │   ├── liquidate.rs
│       │   │   ├── mod.rs
│       │   │   ├── repay.rs
│       │   │   ├── take_loan.rs
│       │   │   └── take_mortgage.rs
│       │   ├── market/
│       │   │   ├── buy_loan.rs
│       │   │   ├── mod.rs
│       │   │   └── sell_loan.rs
│       │   ├── pool/
│       │   │   ├── close_pool.rs
│       │   │   ├── create_pool.rs
│       │   │   ├── liquidity.rs
│       │   │   ├── mod.rs
│       │   │   └── update_pool.rs
│       │   ├── request/
│       │   │   ├── accept_proposal.rs
│       │   │   ├── cancel_proposal.rs
│       │   │   ├── execute_loan.rs
│       │   │   ├── execute_mortgage.rs
│       │   │   ├── mod.rs
│       │   │   └── propose_loan.rs
│       │   ├── state/
│       │   │   ├── collection.rs
│       │   │   ├── loan.rs
│       │   │   ├── mod.rs
```

Files included in the code review
```
│       │   │   ├── pool.rs
│       │   │   ├── request.rs
│       │   │   └── user.rs
│       │   ├── user/
│       │   │   ├── create_stats.rs
│       │   │   ├── mod.rs
│       │   │   └── update_stats.rs
│       │   ├── clean.rs
│       │   ├── error.rs
│       │   ├── expire.rs
│       │   ├── lib.rs
│       │   ├── loan_logic.rs
│       │   ├── mortgage_logic.rs
│       │   ├── pnft.rs
│       │   └── util.rs
│       ├── Cargo.toml
│       └── Xargo.toml
│
```

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

During the Security Assessment of the Rain Program, we discovered:

- 1 finding with LOW severity rating.
- 2 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

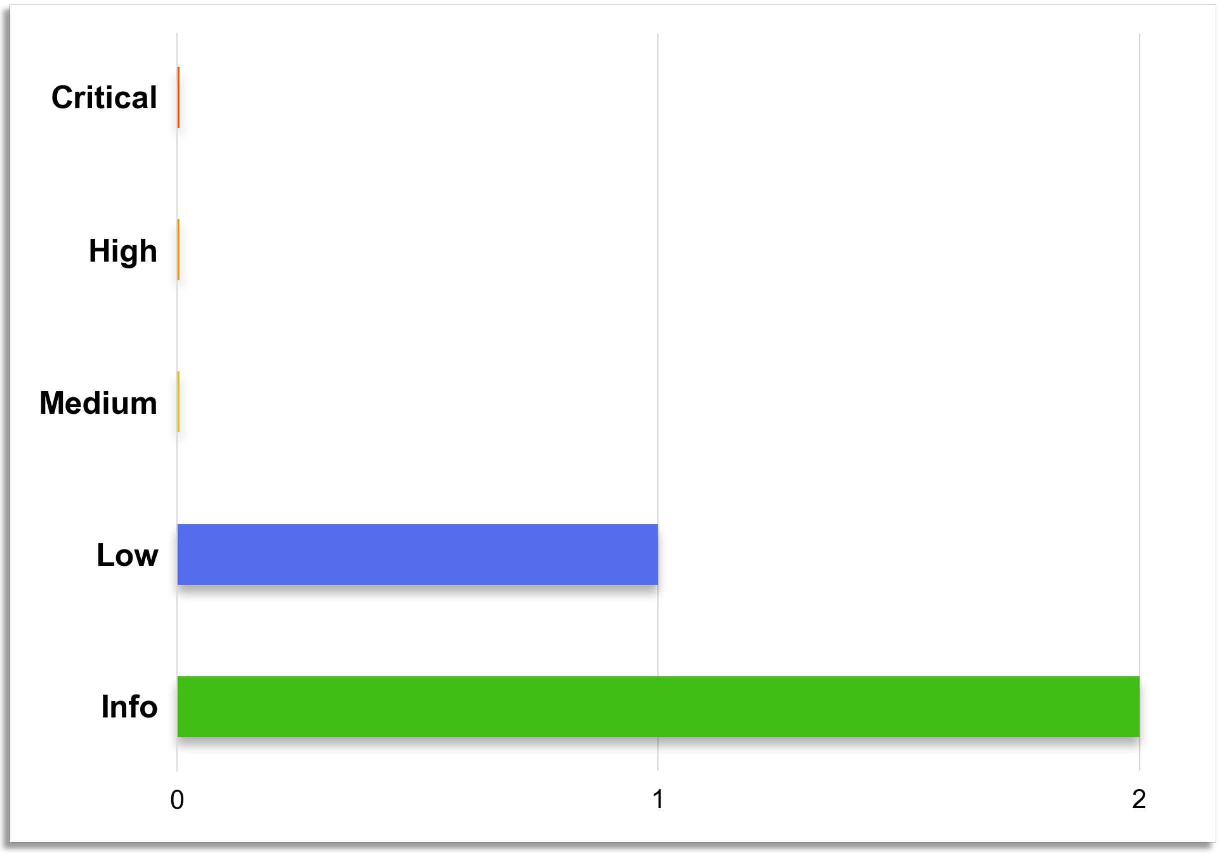

Figure 1: Findings by Severity

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-RF-01	Low	Use-after-free due to a lifetime error in Vec::into_iter()
FYEO-RF-02	Informational	Casting Integer literal to 'u128' is unnecessary
FYEO-RF-06	Informational	Unnecessary equality check against true

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.

Technical Findings

General Observations

During code assessment, it was noted that the rust code is very well written and structured. The use of checked arithmetic operations to protect from overflow/underflow operations shows commitment to writing secure programs. The code documentation is excellent and expansive across the Solana program and SDK. The program strongly relies on ‘anchors’ inbuilt validation and access restriction macros.

It was noted that the account checks done are very concise and show an in-depth understanding of Solana.

Important checks constraints are handled by integrations such as the Cardinal Program Manager and Pyth, which should be continuously monitored for updates or breaking changes to ensure the security posture of the RainFi program.

USE-AFTER-FREE DUE TO A LIFETIME ERROR IN VEC::INTO_ITER()

Finding ID: FYEO-RF-01
Severity: Low
Status: Remediated

Description

In affected versions of this crate, the lifetime of the iterator produced by `Vec::into_iter()` is not constrained to the lifetime of the `Bump` that allocated the vector's memory. Using the iterator after the `Bump` is dropped causes **use-after-free accesses**.

Proof of Issue

File name: Cargo.lock

Line number: 46

```
[[package]]
name = "bumpalo"
version = "3.11.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "c1ad822118d20d2c234f427000d5acc36eabe1e29a348c89b63dd60b13f28e5d"

rain-program/Cargo.lock:46:1
   │
46 │  bumpalo 3.11.0 registry+https://github.com/rust-lang/crates.io-index
   │  ------------------------------------------------------------------
unsound advisory detected
   │
   = ID: RUSTSEC-2022-0078

programs/rain/src/loan/marketplace/solanart.rs
16:    let mut remaining_accounts = ctx.remaining_accounts.into_iter();
87:        .into_iter()

programs/rain/src/loan/marketplace/hadeswap.rs
18:    let mut remaining_accounts = ctx.remaining_accounts.into_iter();
120:        .into_iter()

programs/rain/src/loan/marketplace/auction_house.rs
130:    let mut remaining_accounts = ctx.remaining_accounts.into_iter();
187:            .into_iter()
288:            .into_iter()

programs/global_offers/src/lib.rs
1040:        let mut v_other: Vec<_> = other.into_iter().collect();
1042:        for e1 in self.into_iter() {
1054:        let mut v_other: Vec<_> = other.into_iter().collect();
1056:        for e1 in self.into_iter() {
```

Severity and Impact Summary

In this version dependency, the code gets its underlying Bump’s lifetime threaded through. This meant that rust will not be checking the borrows for `bumpalo::collections::IntoIter` and this could result in use-after-free bugs. ```(ctx.remaining_accounts). remaining_accounts```` is a vector that contains all accounts that were passed into the instruction but are not declared in the Accounts struct.

Recommendation

Upgrade to `>=3.11.1`

https://github.com/fitzgen/bumpalo/blob/main/CHANGELOG.md#3111

CASTING INTEGER LITERAL TO 'U128' IS UNNECESSARY

Finding ID: FYEO-RF-02
Severity: Informational
Status: Remediated

Description

Casting an integer literal to `u128` is unnecessary.

Proof of Issue

File name: programs/rain/src/state/pool.rs

Line number: 90

```
        for i in 1..11 {
            factorial *= i;
            approx_exp += x_powered / factorial;
            x_powered = x_powered * x / INTEREST_PRECISION;
        }

        let base_interest_precise = self.base_interest as u128 *
INTEREST_PRECISION / 100;
        let interest =
            base_interest_precise.max(curve_rate as u128 * approx_exp /
INTEREST_PRECISION);

        (interest * loan_amount as u128 / 100 as u128 / INTEREST_PRECISION)
            .try_into()
            .unwrap()
    }
}
```

Severity and Impact Summary

No security considerations. This helps clarify the intention of the code and reduces complexity.

Recommendation

Remove unnecessary code unless it affects development formatting decisions.

UNNECESSARY EQUALITY CHECK AGAINST TRUE

Finding ID: FYEO-RF-06
Severity: Informational
Status: Remediated

Description

Equality checks against `true` are unnecessary.

Proof of Issue

File name: programs/rain/src/util.rs

programs/rain/src/market/buy_loan.rs

Line number: 171, 234

```
 // Even if mint is manually whitelisted, ensure collection or creator is
still correct
// Collection often update metadatas of stolen NFTs, our whitelist account
could be outdated
        if collection.collection == Pubkey::default() {
            let creators = metadata.data.creators.as_ref().unwrap();
            let matched = creators
                .iter()
                .any(|x| x.address == collection.creator && x.verified ==
true);
            require!(matched, ProgramError::WrongCreatorOrCollection);
        } else {
            let collection_metadata = metadata
                .collection
                .clone()
                .ok_or(ProgramError::WrongCreatorOrCollection)?;

   #[account(
        mut,
        constraint = old_loan.borrower == *borrower.key,
        constraint = old_loan.mint == nft_mint.key(),
        constraint = old_loan.status == LoanStatus::Ongoing,
        constraint = old_loan.pool == pool.key(),
        **constraint = old_loan.sale.is_for_sale == true**,
    )]
    pub old_loan: Box<Account<'info, Loan>>,
```

Severity and Impact Summary

Unnecessary code should be avoided.

Recommendation

Try simplifying it as shown: `old_loan.sale.is_for_sale` and `x.verified`

OUR PROCESS

METHODOLOGY

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

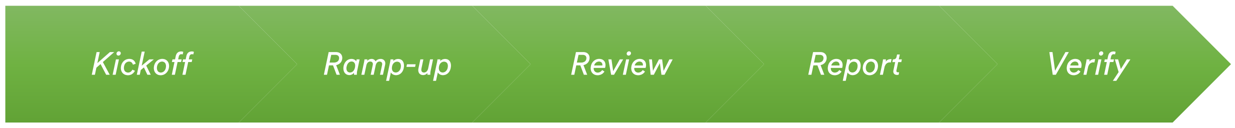

Figure 2: Methodology Flow

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