FYEO

Security Assessment of Bitgreen Carbon Credits

Bitgreen

September 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

Bitgreen engaged FYEO Inc. to perform a Security Assessment of Bitgreen Carbon Credits.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on August 14 - September 19, 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-BGGW-ID-01 – A project can be approved many times, creating copies of each asset
- FYEO-BGGW-ID-02 – BoundedVec will eventually stop function from working
- FYEO-BGGW-ID-03 – It can be impossible to remove authorized users due to binary search
- FYEO-BGGW-ID-04 – Number of validations can be set to 0
- FYEO-BGGW-ID-05 – Return values are ignored
- FYEO-BGGW-ID-06 – Retire data is created for batches with 0 amount to retire
- FYEO-BGGW-ID-07 – Unnecessary mutability

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 Bitgreen Carbon Credits. 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 public repository at https://github.com/bitgreen/bitgreen-node/ with the commit hash master.

Files included in the code review
bitgreen-node |-pallets | |-carbon-credits | | |-src | | | |-types.rs | | | |-lib.rs | | | |-functions.rs | | | |-benchmarking.rs | | | |-mock.rs | | | |-weights.rs | | | |-migration.rs | | | |-tests.rs | |-dex | | |-src | | | |-types.rs | | | |-lib.rs | | | |-benchmarking.rs | | | |-mock.rs | | | |-weights.rs | | | |-tests.rs

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

During the Security Assessment of Bitgreen Carbon Credits, we discovered:

- 1 finding with HIGH severity rating.
- 4 findings with MEDIUM severity rating.
- 2 findings with LOW severity rating.

The following chart displays the findings by severity.

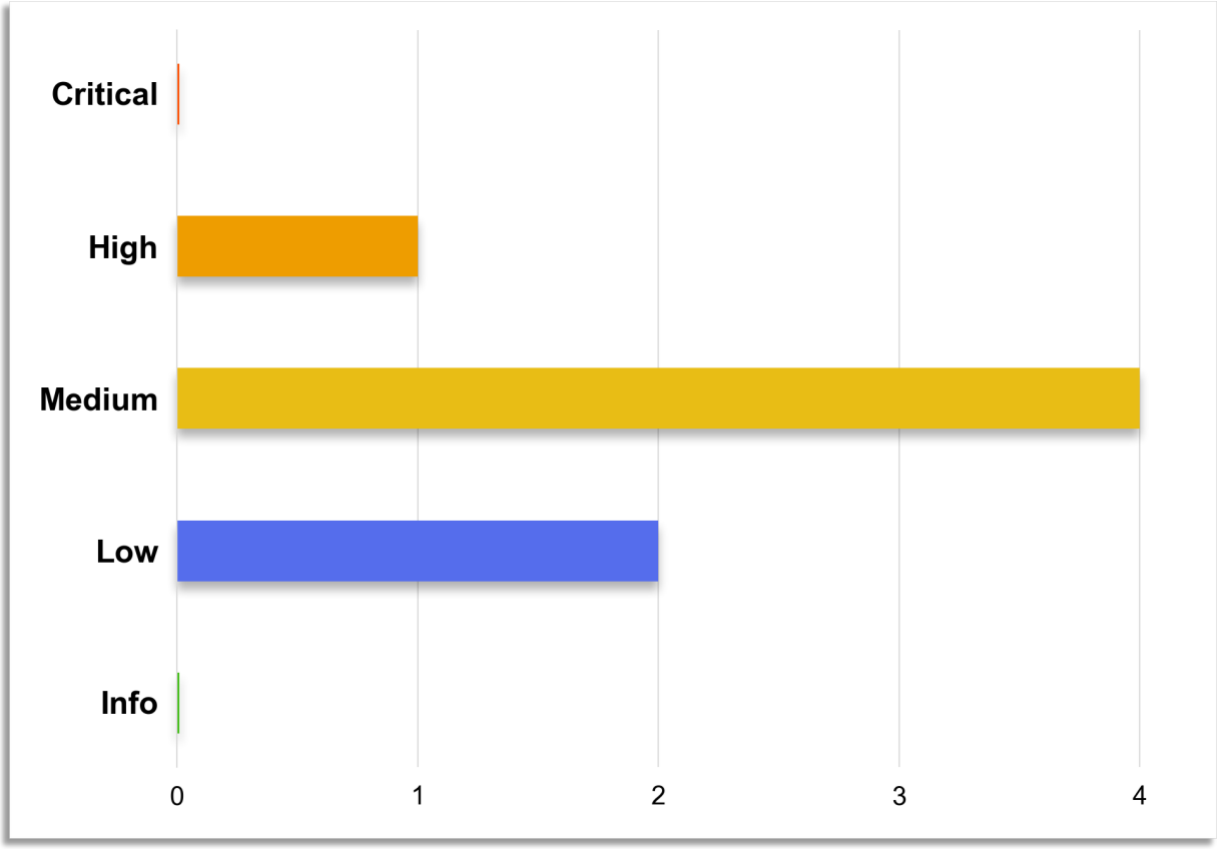


Figure 1: Findings by Severity

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-BGGW-ID-01	**High**	A project can be approved many times, creating copies of each asset
FYEO-BGGW-ID-02	**Medium**	BoundedVec will eventually stop function from working
FYEO-BGGW-ID-03	**Medium**	It can be impossible to remove authorized users due to binary search
FYEO-BGGW-ID-04	**Medium**	Number of validations can be set to 0
FYEO-BGGW-ID-05	**Medium**	Return values are ignored
FYEO-BGGW-ID-06	**Low**	Retire data is created for batches with 0 amount to retire
FYEO-BGGW-ID-07	**Low**	Unnecessary mutability

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

Based on the thorough code review conducted, it is evident that the team has invested significant effort in creating a well-organized code base that is easy to follow. We were happy to find good documentation and a well-laid-out code base during the review of this project.

The team demonstrated great responsiveness and communication skills throughout the process, promptly addressing any questions that arose.

However, it's worth noting that some functions in the code exhibit excessive indentations and length, which can hinder readability. A restructuring effort could greatly improve the overall readability of these functions.

A PROJECT CAN BE APPROVED MANY TIMES, CREATING COPIES OF EACH ASSET

Finding ID: FYEO-BGGW-ID-01
Severity: **High**
Status: **Remediated**

Description

There is no check if a project was already approved. Projects can therefore be approved and unapproved many times.

Proof of Issue

File name: pallets/carbon-credits/src/functions.rs
Line number: 59

```
pub fn do_approve_project(project_id: T::ProjectId, is_approved: bool) ->
DispatchResult {
    Projects::<T>::try_mutate(project_id, |project| -> DispatchResult {
        // ensure the project exists
        let project = project.as_mut().ok_or(Error::<T>::ProjectNotFound)?;

        project.approved = is_approved;

        // if approved, create assets
        if is_approved {
            let mut created_asset_ids: Vec<T::AssetId> = Default::default();

                        for (group_id, mut group) in project.batch_groups.iter_mut() {
                let asset_id = Self::next_asset_id();
                let next_asset_id =
                    asset_id.checked_add(&1u32.into()).ok_or(Error::<T>::Overflow)?;
                NextAssetId::<T>::put(next_asset_id);
```

Severity and Impact Summary

If someone were to approve a project that had been approved before and tokens were already minted and obtained by users, this would cause severe issues as the project would generate all new asset ids when it is approved again. This may lead to a loss of funds for users.

Recommendation

It would be better to have an enum with `Pending`, `Rejected` and `Approved` states or similar. And only execute the approve function if it is in the `Pending` state.

BOUNDEDVEC WILL EVENTUALLY STOP FUNCTION FROM WORKING

Finding ID: FYEO-BGGW-ID-02
Severity: **Medium**
Status: **Remediated**

Description

The `record_payment_to_seller` function will stop working once the `BoundedVec` for `payouts` fills up.

Proof of Issue

File name: pallets/dex/src/lib.rs
Line number: 1003

```
SellerPayouts::<T>::try_mutate(seller.clone(), |payouts| -> DispatchResult {
    let payouts = payouts.get_or_insert_with(Default::default);
    payouts.try_push(payout.clone()).map_err(|_| Error::<T>::PaymentsListFull)?;
    Ok(())
})?;
```

Severity and Impact Summary

This function will stop working once the limit for this `BoundedVec` is reached, as every attempt to insert will cause an error.

Recommendation

Implement some functionality to deal with this situation.

IT CAN BE IMPOSSIBLE TO REMOVE AUTHORIZED USERS DUE TO BINARY SEARCH

Finding ID: FYEO-BGGW-ID-03
Severity: **Medium**
Status: **Remediated**

Description

In `force_remove_authorized_account` and `force_remove_validator_account` there are binary searches used. The addresses are however not inserted in order. This will make it impossible to revoke authorized users properly.

Proof of Issue

File name: pallets/carbon-credits/src/lib.rs
Line number: 462

```
account_list
    .try_push(account_id.clone())
    .map_err(|_| Error::<T>::TooManyAuthorizedAccounts)?;
```

Line number: 481

```
// remove the account_id from the list of authorized accounts if already exists
AuthorizedAccounts::<T>::try_mutate(|account_list| -> DispatchResult {
    if let Ok(index) = account_list.binary_search(&account_id) {
        account_list.swap_remove(index);
        Self::deposit_event(Event::AuthorizedAccountRemoved { account_id });
    }

    Ok(())
})
```

File name: pallets/dex/src/lib.rs
Line number: 853

```
ValidatorAccounts::<T>::try_mutate(|account_list| -> DispatchResult {
    if let Ok(index) = account_list.binary_search(&account_id) {
        account_list.swap_remove(index);
        Self::deposit_event(Event::ValidatorAccountRemoved { account_id });
    }

    Ok(())
})
```

In the dex pallet the same issue is present for `force_remove_validator_account`.

Severity and Impact Summary

Since the items are not ordered during insertion, it can easily become impossible to remove certain users, as a binary search on unsorted data is not going to work as intended.

Recommendation

The order has to be maintained during insertion. Here is an example code:

```
let pos = vec
  .binary_search(&element)
  .err()
  .ok_or(Error::<T>::AlreadyAdded)?;
vec
  .try_insert(pos, element.clone())
  .map_err(|_| Error::<T>::BoundedVecFull)?;
```

NUMBER OF VALIDATIONS CAN BE SET TO 0

Finding ID: FYEO-BGGW-ID-04
Severity: **Medium**
Status: **Remediated**

Description

The number of validations `MinPaymentValidations` can be set to 0.

Proof of Issue

File name: pallets/dex/src/lib.rs
Line number: 866

```
pub fn force_set_min_validations(
    origin: OriginFor<T>,
    min_validators: u32,
) -> DispatchResult {
    T::ForceOrigin::ensure_origin(origin)?;
    MinPaymentValidations::<T>::set(min_validators);
    Ok(())
}
```

Severity and Impact Summary

The `validate_buy_order` function still requires one authorized sender to send their validation data, but with possible future changes it seems important to enforce some minimum threshold.

Recommendation

Enforce a minimum threshold.

RETURN VALUES ARE IGNORED

Finding ID: FYEO-BGGW-ID-05
Severity: **Medium**
Status: **Remediated**

Description

The return values of some function calls are ignored.

Proof of Issue

File name: pallets/dex/src/lib.rs
Line number: 419

```
BuyOrdersByUser::<T>::try_mutate(
    buy_order.buyer.clone(),
    |open_orders| -> DispatchResult {
        let mut open_orders = open_orders.get_or_insert_with(Default::default);
        open_orders.retain(|&x| x != (key, buy_order.units));
        Ok(())
    },
);
```

Severity and Impact Summary

The error has to be handled. If for whatever reason this call fails, the `BuyOrdersByUser` data will be corrupted.

Recommendation

Handle the error.

RETIRE DATA IS CREATED FOR BATCHES WITH 0 AMOUNT TO RETIRE

Finding ID: FYEO-BGGW-ID-06
Severity: **Low**
Status: **Remediated**

Description

Retire data is created even if some batch has a `0` actual amount.

Proof of Issue

File name: pallets/carbon-credits/src/functions.rs
Line number: 530

```
for batch in batch_list.iter_mut() {
    // lets retire from the older batches as much as possible
    // this is safe since we ensure minted >= retired
    let available_to_retire =
        batch.minted.checked_sub(&batch.retired).ok_or(Error::<T>::Overflow)?;

    let actual = cmp::min(available_to_retire, remaining);

    batch.retired = batch.retired.checked_add(&actual).ok_or(Error::<T>::Overflow)?;

    // create data of retired batch
    let batch_retire_data: BatchRetireDataOf<T> = BatchRetireData {
        name: batch.name.clone(),
        uuid: batch.uuid.clone(),
        issuance_year: batch.issuance_year,
        count: actual,
    };
```

Severity and Impact Summary

Not really a security related issue. It will however store useless data on chain.

Recommendation

Skip over empty batches.

UNNECESSARY MUTABILITY

Finding ID: FYEO-BGGW-ID-07
Severity: **Low**
Status: **Remediated**

Description

Unnecessary mutability of some variables.

Proof of Issue

File name: pallets/dex/src/lib.rs
Line number: 422

```
let mut open_orders = open_orders.get_or_insert_with(Default::default);
```

Line number: 762

```
let mut open_orders =
```

Severity and Impact Summary

No security impact. But it is better to not allow for unnecessary mutability.

Recommendation

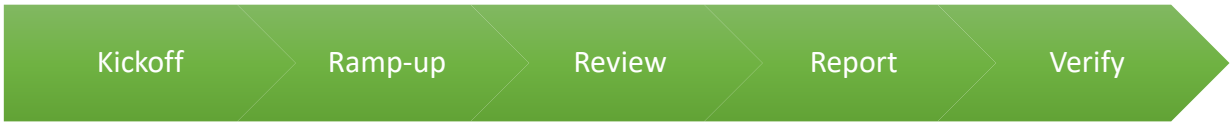
Remove the `mut` when not needed.

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