FYEO

Security Assessment of the Bitgreen Node

Bitgreen, Inc.

September 2022
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, Inc. engaged FYEO Inc. to perform a Security Assessment of the Bitgreen Node.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on August 04 - August 22, 2022, 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 were prioritized for remediation to reduce to the risk they pose:

- FYEO-BG-01 – Using transfer with keep_alive keeps VCU tokens stuck
- FYEO-BG-02 – Create pool must be expensive
- FYEO-BG-03 – Deposit can be run with amount 0
- FYEO-BG-04 – Retire can be run with amount 0; Will create an NFT
- FYEO-BG-05 – Individual batches could have a 0 amount
- FYEO-BG-06 – Possible overflows
- FYEO-BG-07 – Retired tokens should be equal or less than minted
- FYEO-BG-08 – Use of expect()
- FYEO-BG-09 – Mint can be run with amount 0
- FYEO-BG-10 – No way to cleanup an asset if a project is not approved
- FYEO-BG-11 – Resubmit does not check batch_total_supply > 0
- FYEO-BG-12 – The year is stored as a u32

Based on our review, we conclude that the reviewed code implements the documented functionality and that all findings have been sufficiently remediated.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Assessment of the Bitgreen Node. 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/tree/release with the commit hash dc5e4bacd2a7936f15bb5afdbdde2e06d4826861.

A re-review was performed on September 5, 2022, with the commit hash
2c91ba70a935fb89a7372dfb026e74796821a9ea

Files included in the code review

```
node/
├── pallets/
│   ├── vcu/
│   │   ├── src/
│   │   │   ├── benchmarking.rs
│   │   │   ├── functions.rs
│   │   │   ├── lib.rs
│   │   │   ├── mock.rs
│   │   │   ├── tests.rs
│   │   │   ├── types.rs
│   │   │   └── weights.rs
│   │   └── Cargo.toml
│   └── vcu-pools/
│       ├── src/
│       │   ├── benchmarking.rs
│       │   ├── lib.rs
│       │   ├── mock.rs
│       │   ├── tests.rs
│       │   ├── types.rs
│       │   └── weights.rs
│       └── Cargo.toml
├── parachain/
│   ├── src/
│   │   ├── chain_spec.rs
│   │   ├── cli.rs
│   │   ├── command.rs
│   │   ├── main.rs
│   │   ├── rpc.rs
│   │   └── service.rs
│   ├── Cargo.toml
│   └── build.rs
├── primitives/
```

Files included in the code review

```
│    ├── src/
│    │    ├── lib.rs
│    │    └── vcu.rs
│    └── Cargo.toml
├── runtime/
│    └── parachain/
│         ├── src/
│         │    ├── benchmarking/
│         │    │    ├── authority.rs
│         │    │    ├── evm.rs
│         │    │    ├── evm_accounts.rs
│         │    │    ├── mod.rs
│         │    │    └── utils.rs
│         │    ├── weights/
│         │    │    ├── block_weights.rs
│         │    │    ├── evm.rs
│         │    │    ├── evm_accounts.rs
│         │    │    ├── extrinsic_weights.rs
│         │    │    ├── mod.rs
│         │    │    ├── paritydb_weights.rs
│         │    │    ├── rocksdb_weights.rs
│         │    │    └── transaction_payment.rs
│         │    ├── lib.rs
│         │    └── xcm_config.rs
│         ├── Cargo.toml
│         └── build.rs
├── Cargo.lock
└── Cargo.toml
```

Table 1: Scope

Technical Analyses and Findings

During the Security Assessment of the Bitgreen Node, we discovered:

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

The following chart displays the findings by severity.

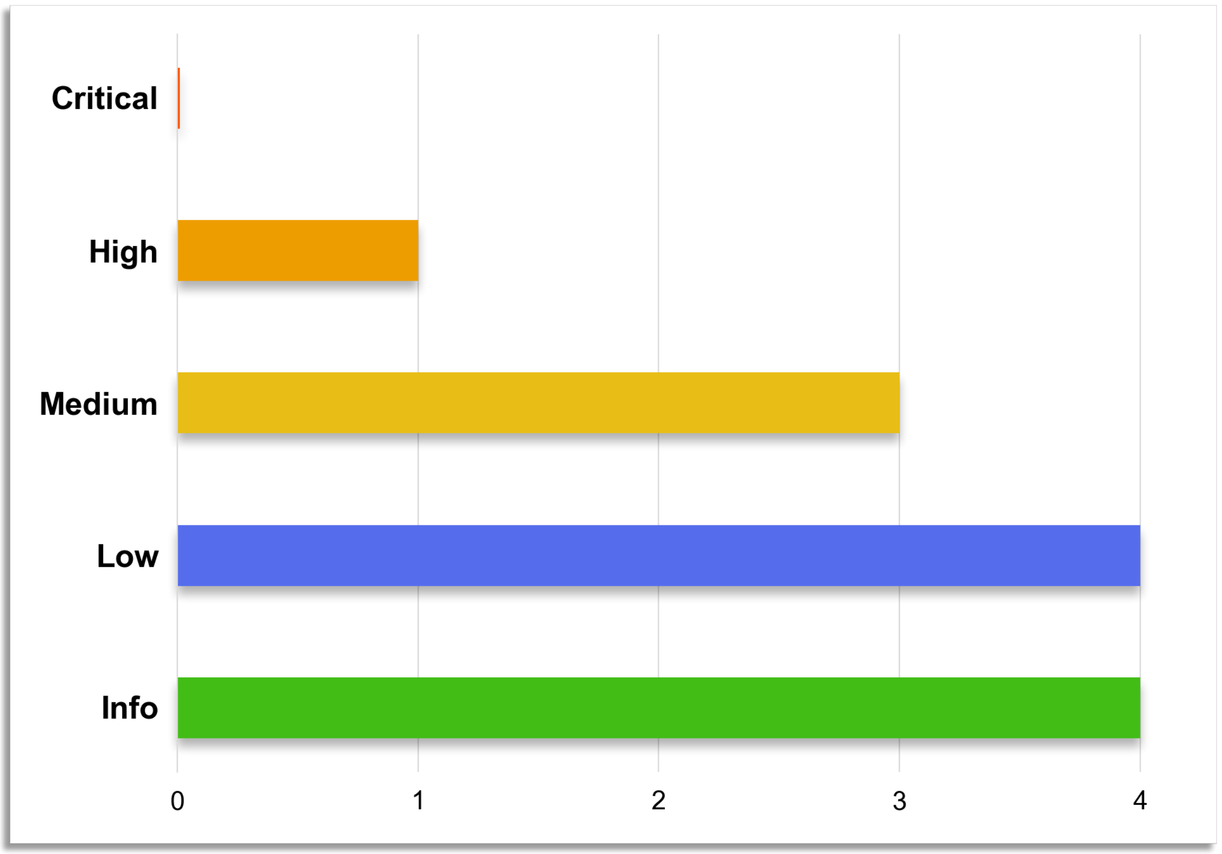


Figure 1: Findings by Severity

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-BG-01	**High**	Using transfer with keep_alive keeps VCU tokens stuck
FYEO-BG-02	**Medium**	Create pool must be expensive
FYEO-BG-03	**Medium**	Deposit can be run with amount 0
FYEO-BG-04	**Medium**	Retire can be run with amount 0; Will create an NFT
FYEO-BG-05	**Low**	Individual batches could have a 0 amount
FYEO-BG-06	**Low**	Possible overflows
FYEO-BG-07	**Low**	Retired tokens should be equal or less than minted
FYEO-BG-08	**Low**	Use of expect()
FYEO-BG-09	**Informational**	Mint can be run with amount 0
FYEO-BG-10	**Informational**	No way to cleanup an asset if a project is not approved
FYEO-BG-11	**Informational**	Resubmit does not check batch_total_supply > 0
FYEO-BG-12	**Informational**	The year is stored as a u32

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

During code assessment it was noted that the rust code is 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 good; however, the additional documentation in the doc folder is unfortunately outdated.

The chain relies quite heavily on trusted parties. Projects can be created by accounts added to the KYC membership pallet and then the project needs to be approved by an authorized account. The sudo pallet is used to configure authorized accounts.
Once approved, one of the authorized accounts can issue (mint) VCU tokens.

The admin (sudo pallet) has substantial powers in this project. Functions exist which, among other things, can be used to overwrite project and pool data as well as NFT meta data at will. These functions do not check basic data integrity and thereby create a risk of data corruption should the admin make a mistake. This is also a risk in case keys get compromised or if someone with access to admin keys decides to behave maliciously.

USING TRANSFER WITH KEEP_ALIVE KEEPS VCU TOKENS STUCK

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

Description

When using the `transfer()` function with `keep_alive` set to true, the minimum deposit will be enforced so the account won't be closed.

Proof of Issue

File name: pallets/vcu-pools/src/lib.rs
Line number: 391

```
<T as pallet::Config>::AssetHandler::transfer(
    *project_id,
    &Self::account_id(),
    &who,
    actual,
    true,
)?;
```

File name: pallets/vcu-pools/src/lib.rs
Line number: 290

```
<T as pallet::Config>::AssetHandler::transfer(
    project_id,
    &who,
    &Self::account_id(),
    amount,
    true,
)?;
```

Severity and Impact Summary

A VCU pool can store VCU tokens for a variety of different projects. When users deposit any of the accepted VCU tokens, they receive a new pool token at a 1:1 ratio.

Users can then retire the pool tokens, which will then transfer VCU tokens, based on a oldest batch (issuance year) first approach, from the pool to the user and then call the VCU retire function on these. As the function uses `keep_alive`, it becomes impossible to transfer out the last token of any project. It is therefore possible that certain pools become unable to retire tokens.

For instance, if a pool's oldest batch has the last remaining tokens of some project but newer batches of a variety of others, it will be impossible to retire any of the new batches as well, since 1 token of the oldest batch will forever remain in the pool and the transfer function will fail the whole retire instruction right away.

As pools are public, someone holding a majority of remaining tokens for a project that no longer issues new ones, could also use this to block a pools newer batches if that pool accepts tokens of that particular project.

It would then be required to issue new VCU tokens for that particular project and put them into some VCU pool. But if the project does no longer issue new tokens, it would require `sudo` intervention.

Recommendation

The transfer function should not keep accounts alive when retiring tokens. The deposit function should also allow people to deposit all their tokens into the pool.

CREATE POOL MUST BE EXPENSIVE

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

Description

Both the VCU and VCU pool pallet share the same asset id space which is a `u32`. Create pool is a permissionless function and unless the transaction fee is prohibitive, it may be possible for a user to create billions of pools until there are no more free asset ids available.

Each pool also stores a considerable amount of on-chain data.

Severity and Impact Summary

In the case where the transaction fee is too low, it would be possible for a user to store a lot of useless pool data on-chain which would increase validator requirements. In the worst case, all possible `u32` asset ids may be blocked out.

Recommendation

A possible solution would require a minimum deposit when creating a pool. Perhaps x amount of VCUs could be burned into an NFT for the creator.

Deposit can be run with amount 0

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

Description

The VCU pool deposit function can be called with `amount = 0`. It will still store on-chain data.

Proof of Issue

The asset `transfer()` function will work with amount 0 and return `Ok()`.

```
if amount.is_zero() {
    return Ok((amount, None))
}
```

Severity and Impact Summary

The function will store some on-chain data without any need.

Recommendation

Check that the amount is `> 0`.

Retire can be run with amount 0; Will create an NFT

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

Description

The `retire()` functions on both the VCU and VCU pool pallet can be run with `amount = 0`. The user will still receive an NFT and the associated meta data will be stored on-chain.

Proof of Issue

The `burn_from()` function will return `Ok()` for an amount of 0.

```
if amount.is_zero() {
    return Ok(amount)
}
```

Severity and Impact Summary

This function will create an NFT for the user and store meta data on-chain. However, the user didn't actually retire anything. Of course a transaction fee still has to be paid.

Recommendation

Make sure the amount is at least `1`.

INDIVIDUAL BATCHES COULD HAVE A 0 AMOUNT

Finding ID: FYEO-BG-05
Severity: **Low**
Status: **Remediated**

Description

An individual batch could be `0`, thus storing useless data on chain. This calculation is used in both `create_project()` and `resubmit_project()`.

Proof of Issue

File name: pallets/vcu/src/functions.rs
Line number: 96

```
let batch_total_supply =
    params
        .batches
        .iter()
        .fold(Zero::zero(), |mut sum: T::Balance, batch| {
            sum += batch.total_supply;
            sum
        });
```

Severity and Impact Summary

An admin is required to verify this data. Still a sanity check is useful.

Recommendation

Return an error if the amount of an individual batch is 0. The project can always be re-submitted if it had issues.

POSSIBLE OVERFLOWS

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

Description

There are some possible overflows that should be addressed.

Proof of Issue

File name: pallets/vcu/src/functions.rs
Line number: 241

```
ensure!(
    amount_to_mint + project.minted <= project.total_supply,
    Error::<T>::AmountGreaterThanSupply
);
```

> In `mint_vcus()`, the check can be bypassed with an overflow. The variable `amount_to_mint` is a user supplied parameter. However, there are checks in place that make sure the function does not have remaining tokens after creating its batches.

File name: pallets/vcu/src/functions.rs
Line number: 257

```
let available_to_mint = batch.total_supply - batch.minted;
```

> In `mint_vcus()` this underflow is generally not possible, as minted should never exceed total_supply, however admins can modify all data and might make a mistake thereby breaking this assumption. A similar issue with `retire_vcus()` exists with `batch.minted - batch.retired`.

File name: pallets/vcu/src/functions.rs
Line number: 266, 367

```
remaining -= actual;
```

> In `mint_vcus()` this underflow is generally not possible, however if admins made mistakes modifying the data, the operations above can underflow which can, in turn, underflow this operation. The same applies to `retire_vcus()`.

File name: pallets/vcu-pools/src/lib.rs
Line number: 308

```
let new_amount = *existing_amount + amount;
```

> In the pool deposit() function. It is unlikely to overflow as the balance is a u128 and 1 VCU is expected to be prohibitively expensive and the projects are approved by admins. However, admins could make mistakes.

File name: pallets/vcu/src/functions.rs
Line number: 101

```
let batch_total_supply =
    params
        .batches
        .iter()
        .fold(Zero::zero(), |mut sum: T::Balance, batch| {
            sum += batch.total_supply;
            sum
        });
```

> In retire_vcus() and resubmit_project() the calculation of totals can overflow. However, admins are required to approve of these data.

Severity and Impact Summary

Some of the overflows are only possible if the admin made a mistake when editing data. But since the admin can write anything, it would not hurt to have these overflow checks.

Recommendation

Use the appropriate checked math functions.

Retired tokens should be equal or less than minted

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

Description

The number of retired tokens is verified to be <= the total supply. However, this should be compared against the minted supply, as only minted tokens can be retired.

Proof of Issue

File name: pallets/vcu/src/functions.rs
Line number: 331

```
ensure!(
    project.retired <= project.total_supply,
    Error::<T>::AmountGreaterThanSupply
);
```

Severity and Impact Summary

This will not have a serious impact, as there are more tests at the end of the instruction that make sure the numbers are in order.

Recommendation

Check against the number of minted tokens.

USE OF EXPECT()

Finding ID: FYEO-BG-08
Severity: **Low**
Status: **Remediated**

Description

The use of `.expect()` should be avoided in the substrate runtime - an error should be returned.

Proof of Issue

File name: pallets/vcu/src/functions.rs
Line number: 279

```
project.batches = batch_list
    .try_into()
    .expect("This should not fail since we did not change the size. qed");
```

File name: pallets/vcu/src/functions.rs
Line number: 390

```
project.batches = batch_list
    .try_into()
    .expect("This should not fail since the size is unchanged. qed");
```

Severity and Impact Summary

While this should not fail, returning an error is the more appropriate solution.

Recommendation

Return an error instead of causing a panic.

References

https://github.com/paritytech/substrate/issues/130

Mint can be run with amount 0

Finding ID: FYEO-BG-09
Severity: **Informational**
Status: **Remediated**

Description

The VCU mint function can be called with `amount = 0`.

Proof of Issue

The asset `mint_into()` function will work with amount 0 and return `Ok()`.

```
if amount.is_zero() {
    return Ok(())
}
```

Severity and Impact Summary

The user would be allowed to pay a fee to do nothing. This function does also requires authorization.

Recommendation

Check that the amount is `> 0`.

NO WAY TO CLEANUP AN ASSET IF A PROJECT IS NOT APPROVED

Finding ID: FYEO-BG-10
Severity: **Informational**
Status: **Remediated**

Description

The `create_project()` function creates a new token asset when the project is setup. There is, however, no guarantee that the project will ever be approved. There is also no function to release this data.

Severity and Impact Summary

Neither the project nor its asset can be removed in case that project approval will never be given.

Recommendation

Consider if such functionality is desirable or not.

RESUBMIT DOES NOT CHECK BATCH_TOTAL_SUPPLY > 0

Finding ID: FYEO-BG-11
Severity: **Informational**
Status: **Remediated**

Description

The `create_project()` function has a check that `batch_total_supply > Zero::zero()`. This check was not added to `resubmit_project()`.

Severity and Impact Summary

Project data ultimately requires approval, but the test would pick up on user mistakes. It is recommended to have the test in both functions.

Recommendation

Add the check to `resubmit_project()`.

THE YEAR IS STORED AS A u32

Finding ID: FYEO-BG-12
Severity: **Informational**
Status: **Remediated**

Description

There is no need to store the year as a `u32`. A `u16` would suffice until the year 65535.

Proof of Issue

File name: primitives/src/vcu.rs
Line number: 4

```
pub type IssuanceYear = u32;
```

Severity and Impact Summary

While not much of a concern, it would still save a tiny bit of storage across all data structures using it.

Recommendation

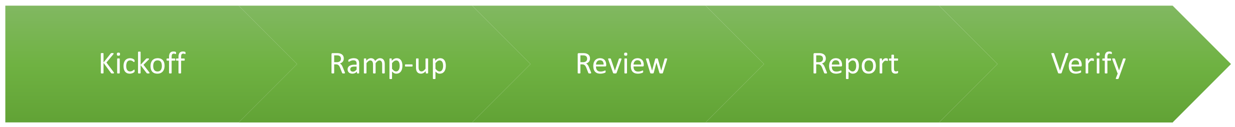
Consider using a `u16`.

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