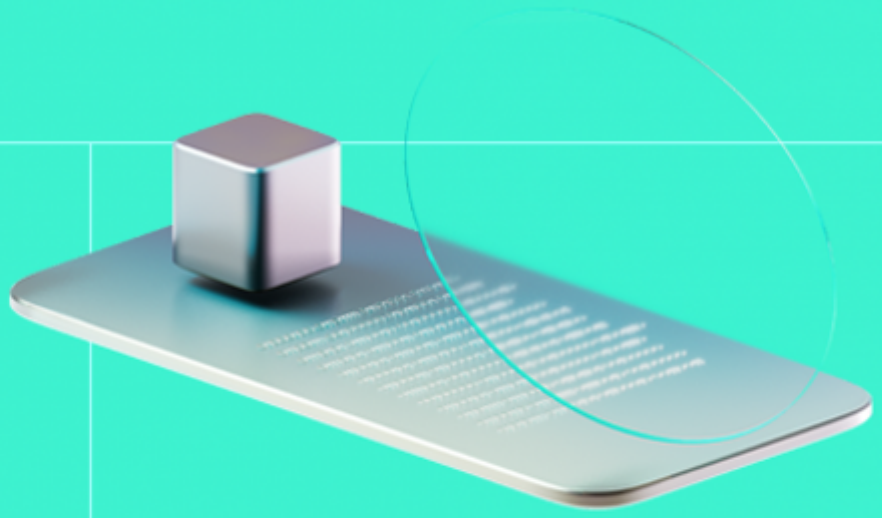
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Gable Finance

Date: 01/11/2023

We express our gratitude to the Gable Finance team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Gable is a protocol creating the liquidity market on Radix DLT. Users can borrow funds without collateral in the form of flash loans, as well as earn staking rewards and interest earnings from supplying their tokens to the protocol.

Platform: Radix DLT

Language: Rust/Scrypto

Tags: Staking, Flashloans

Timeline: 03/10/2023 - 01/11/2023

Methodology: https://hackenio.cc/sc_methodology

Review

Scope

Repository	https://github.com/gable-finance/gable/tree/main/src/backend/scrypto/flashloan-pool/
Commit	afeb0343533798020630fcf45432abce7580b7e8

Audit Summary

10/10	10/10	66.66%	10/10
Security score	Code quality score	Test coverage	Documentation quality score

Total 8.7/10

The system users should acknowledge all the risks summed up in the risks section of the report

8	8	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	1
Medium	2
Low	5

Vulnerability	Status
F-2023-0151 - Owner could withdraw more than he has deposited as owner_liquidity	Fixed
F-2023-0152 - Missing upper bound on interest rate change	Fixed
F-2023-0153 - Missing validations in multiple calculations could lead to unexpected state	Fixed
F-2023-0154 - Macros used for debugging should not be used in production code	Fixed
F-2023-0155 - Owner is able to unlock and update royalties for function calls	Fixed
F-2023-0156 - Wrong limit for the size of box	Fixed
F-2023-0157 - Floating Language Version	Fixed
F-2023-0158 - Test functions should be removed	Fixed

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Gable Finance
Audited By	Hacken
Website	https://gable.finance/
Changelog	10/10/2023 - Initial Review
	01/11/2023 - Second Review

Table of Contents

System Overview

Gable is a protocol creating the liquidity market on Radix DLT. Users can borrow funds without collateral in the form of flash loans, as well as earn staking rewards and interest earnings from supplying their tokens to the protocol.

Users that are staking XRD on the *Gable* validator receive liquid staking units (LSU tokens) that can be deposited in the *flashloan* pool to earn some interest.

Borrowers that take loans repay it with some interest that is then split 50-50 between the users that deposited LSU and the smart contract owner.

Privileged roles

- The owner of the contract could perform multiple administrative changes, like changing interest rates, updating suppliers key value store, deposit and withdraw liquidity to the pool directly, depositing and withdrawing validator node ownership token, as well as perform unstaking and claiming operations.
- The admin role can perform most of the owner's actions, except from depositing and withdrawing liquidity and validator node ownership token.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation quality score is **10** out of **10**.

- Functional and technical requirements were provided.
- Technical description and diagrams were provided.
- The code implements complex calculations logic with small amounts of descriptions and requirements.

Code quality

The total Code quality score is **10** out of **10**.

- The development environment is configured.
- The code is readable and easy to digest.
- Most of the methods are described with appropriate comments.
- Test cases are well described with requirements.

Test coverage

Code coverage of the project could not be directly calculated with common tools likely due to the lack of support for Scrypto. Nevertheless, taking into account the functional coverage and the number of tests available in the code repository, the tests cover approximately **66.66%** of the codebase.

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is missed.
- Interaction with validator and methods associated with these operation are not covered

Security score

Upon auditing, the code was found to contain **0** critical, **1** high, **2** medium, and **5** low severity issues. Out of these, **8** issues have been addressed and resolved, leading to a Security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **8.7**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- The smart contract could be upgraded and its functionality may be changed.
- Centralization and the owner's ability to withdraw the whole liquidity from the pool might be dangerous, if his wallet/badge will be compromised.

Findings

Vulnerability Details

F-2023-0151 - Owner could withdraw more than he has deposited as owner_liquidity - High

Description:

The owner can provide funds to the flashloan pool in the form of a deposit, so that there is some liquidity in the protocol - as well as withdraw them. The functions `owner_deposit_xrd` and `owner_withdraw_xrd` are used for this purpose.

An incorrect validation was found in the `owner_withdraw_xrd` function regarding the verification on whether the amount paid is equal to or less than the liquidity in the pool. There should be a check comparing the mentioned amount with the amount of liquidity deposited by the owner.

```
pub fn owner_withdraw_xrd(&mut self, amount: Decimal) -> Bucket {
// Ensure amount is positive
assert!(
amount > Decimal::ZERO,
"Please withdraw an amount larger than 0"
);

// Ensure amount is less or equal to liquidity provided by owner
assert!(
amount <= self.liquidity_pool_vault.amount(),
"Please withdraw an amount smaller than or equal to {}",
self.owner_liquidity
);
```

Consequently, if, via the `claim_xrd` function, there are more funds in the pool than the owner initially deposited, then he is able to withdraw all of them, even though they are not his property. This is possible because amounts and liquidity are based on values of the *Decimal* type, which can be negative. Then the contract will not return panic if `owner_liquidity` drops below zero, for example to -1000.

Assets:

- flashloan-pool/src/lib.rs [https://github.com/gable-finance/gable/tree/main/src/backend/scrypto/flashloan-pool/]

Status: Fixed

Classification

Severity: High

Recommendations

Remediation: `owner_withdraw_xrd` function should be adjusted so that assert is checking that the amount requested is equal to or less than the liquidity owned by the owner.

Resolution: The Finding was fixed in commit a3fe638.

Evidences

Proof of Concept

Reproduce:

Test shows that after the validator sends funds to the liquidity pool, the owner is able to withdraw 400 tokens more than they themselves deposited.

```
#[test]
fn owner_withdraws_more_than_deposited() -> Result<(), RuntimeError>
{
//
let (mut env, mut flashloanpool) = setup_flashloan_pool()?;
let xrd_bucket: Bucket = env.with_auth_module_disabled(|env| {
// owner deposits 100 XRD
let rtn = ResourceManager(XRD).mint_fungible(100.into(), env);
let _ = flashloanpool.owner_deposit_xrd(rtn.unwrap(), env);
// validator sends 1000 XRD as staking rewards
let rtn2 = ResourceManager(XRD).mint_fungible(1000.into(), env);
let _ = flashloanpool.deposit_batch(rtn2.unwrap(), env);
// owner withdraws 500 XRD, 400 more than he deposited
let bucket = flashloanpool.owner_withdraw_xrd(dec!("500"), env);
bucket
})?;

let flashloanpool_state = env.read_component_state::<FlashloanpoolSt
ate, _>(flashloanpool)?;
let xrd_amount = flashloanpool_state.liquidity_pool_vault.amount(&mu
t env)?;
let owner_amount = flashloanpool_state.owner_liquidity;
// Tokens left in the pool
assert_eq!(xrd_amount, dec!("600"));
// Tokens owned by the owner - as far as he withdraws more than he d
eposited, value is lower than zero
assert_eq!(owner_amount, dec!("-400"));
Ok(())
}
```

F-2023-0152 - Missing upper bound on interest rate change - Medium

Description: It was noticed that one of the methods available only to the contract *OWNER* or wallet with the *ADMIN* role is to change the interest rate in the protocol. While there is validation that this value is not negative, there is no upper maximum value it can take. As a consequence, the user may pay a "fee" several times higher than the loan amount they took out.

This is especially dangerous in the current configuration, in which *OWNER* is a single wallet and is exposed to compromise.

Assets:

- flashloan-pool/src/lib.rs [https://github.com/gable-finance/gable/tree/main/src/backend/scrypto/flashloan-pool/]

Status: Fixed

Classification

Severity: Medium

Recommendations

Remediation: A maximum cap on the `interest_rate` variable should be implemented so that even the *OWNER* of the contract cannot set it to an illogically high value that affects critical protocol functionality.

Resolution: The Finding was fixed in commit cedf40a.

F-2023-0153 - Missing validations in multiple calculations could lead to unexpected state - Medium

Description:

Most of the numeric variables in the contract are of *Decimal* type. Unlike *Uint*, *Decimal* allows you to store and manipulate negative numbers. If the business logic does not take into account a number less than zero in a given context, appropriate validation should take place so that if the "zero" threshold is exceeded, an error is returned.

This situation can be observed in the `update_aggregate_im` function, where during the `interest_new` calculation it is not verified whether `owner_liquidity` is not equal to `total_liquidity`, so if the values of `rewards_new`, `rewards_aggregated` or `interest_aggregated` are greater than zero, then `interest_new` will be negative.

Additionally, in the `update_supplier_kvs` function it was noticed that there is no validation whether `box_lsu` is different from zero, which may cause the contract to panic when calculating the `supplier_relative_lsu_stake` variable.

Assets:

- flashloan-pool/src/lib.rs [https://github.com/gable-finance/gable/tree/main/src/backend/scrypto/flashloan-pool/]

Status: Fixed

Classification

Severity: Medium

Recommendations

Remediation:

We suggest adding additional validations in all places where *Decimal* values should not exceed the zero threshold. Additionally, the suggested solution is to verify all divisor values in the context of their being different from zero.

Resolution:

The Finding was fixed in commit e5c8e29.

F-2023-0154 - Macros used for debugging should not be used in production code - Low

Description: The current contract code has been found to contain many instances of debugging macros, such as `debug!()` and `info!()`. While they are very helpful during the code development and testing phase, their use in production code is considered bad practice. Additionally, each operation that stores some data in memory causes the virtual machine to perform some work - and thus increases the cost of gas needed to perform this transaction.

Assets:

- flashloan-pool/src/lib.rs [https://github.com/gable-finance/gable/tree/main/src/backend/scrypto/flashloan-pool/]

Status: Fixed

Classification

Severity: Low

Recommendations

Remediation: If this type of methods are used to transfer certain state to off-chain components, for example in the form of logs - it seems more appropriate to use event emitting. Otherwise, you should consider removing unnecessary code fragments from the codebase.

Resolution: The Finding was fixed in commit 6cfbaf4.

F-2023-0155 - Owner is able to unlock and update royalties for function calls - Low

Description:

The owner has set up some royalties for public methods and even specified that the amount is locked. However they reserved a right to unlock and update the amount they charge from users for calling these methods (the protocol enforces a hard cap).

Since the royalties are marked as locked, the assumption is that they should be unchanged.

```
.enable_component_royalties(component_royalties! {
roles {
royalty_setter => OWNER;
royalty_setter_updater => OWNER;
royalty_locker => OWNER;
royalty_locker_updater => OWNER;
royalty_claimer => OWNER;
royalty_claimer_updater => OWNER;
},
init {
get_flashloan => Xrd(1.into()), locked;
...
```

Assets:

- flashloan-pool/src/lib.rs [https://github.com/gable-finance/gable/tree/main/src/backend/scrypto/flashloan-pool/]

Status: Fixed

Classification

Severity: Low

Recommendations

Remediation: Restrict the ability to unlock royalties and/or mark the royalties as *updatable*. You can also specify the amount to be an approximate USD equivalent with `Usd(amount.into())`.

Resolution: The Finding was fixed in commit aee1aca.

F-2023-0156 - Wrong limit for the size of box - Low

Description: The `box_size` is set to *250* in the beginning. It is possible to lower it, but not possible to change it back to *250*. This change multiplies as the `box_size * box_size` is the maximum possible number of users of the contract.

Assets:

- flashloan-pool/src/lib.rs [https://github.com/gable-finance/gable/tree/main/src/backend/scrypto/flashloan-pool/]

Status: Fixed

Classification

Severity: Low

Recommendations

Remediation: Check the value inclusively.

Resolution: The Finding was fixed in commit a748fc7.

F-2023-0157 - Floating Language Version - Low

Description: It is preferable for a production project, especially a smart contract, to have the programming language version pinned explicitly. This results in a stable build output, and guards against unexpected toolchain differences or bugs present in older versions, which could be used to build the project.

The language version could be pinned in automation/CI scripts, as well as proclaimed in README or other kinds of developer documentation. However, in the Rust ecosystem, it can be achieved more ergonomically via a *rust-toolchain.toml* descriptor (see https://rust-lang.github.io/rustup/overrides.html#the-toolchain-file).

Status: Fixed

Classification

Severity: Low

Recommendations

Remediation: Pin the language version at the project level.

Resolution: The Finding was fixed in commit e8da2c1.

F-2023-0158 - Test functions should be removed - Low

Description: One of the methods available for public call is `deposit_batch`. It is used to manually add funds to stacking rewards. However, both in the documentation and in the contract code it is specified as “Temporary” due to the need to simulate the validator during the testing phase.

While it does not pose a direct threat to the contract, such features should not be available in the production version of the code.

Assets:

- flashloan-pool/src/lib.rs [https://github.com/gable-finance/gable/tree/main/src/backend/scrypto/flashloan-pool/]

Status: Fixed

Classification

Severity: Low

Recommendations

Remediation: The final version of the protocol should not contain any test functions or functions simulating specific operations. In the code marked as the 'release version', the `deposit_batch` function and its associated method must be removed.

Resolution: Revised in commits 49452f7 and 53f392b.

Observation Details

I-2023-0038 - Suggestion for searching a vacant box - Info

Description:

In the `deposit_lsu` function, the code is looking for an index of a box (`box_nr`) for saving deposit info.

```
for (key, values) in &self.supplier_aggregate_im {
// Check if the Vec is not empty and satisfies your condition
if let Some(first_value) = values.first() {
if *first_value < self.box_size.into() {

// update box number
box_nr = *key;

// update existing supplier's info before adding a new supplier
self.update_supplier_kvs(box_nr);

// Set the flag to true to indicate that the condition has been sati
sfied
condition_satisfied = true;

break;
}
}
}
```

After that condition is checked, and based on a result, some actions are performed:

```
if condition_satisfied {
// Scenario 1: Add new supplier to the existing key value store and
index map
...
} else {
// Scenario 2: In case that all boxes are full or no box exists, a n
ew box has to be inserted
...
}
```

The code can be simplified to improve Code Quality.

Assets:

- flashloan-pool/src/lib.rs [https://github.com/gable-finance/gable/tree/main/src/backend/scrypto/flashloan-pool/]

Status: Fixed

Recommendations

Remediation:

Consider using optional box index, and matching on the option. Sample implementation is lister below:

```
let mut vacant_box = None;
for (box_nr, values) in &self.supplier_aggregate_im {
if values.first().is_some_and(|suppliers_in_box| *suppliers_in_box <
self.box_size.into()) {
vacant_box = Some(*box_nr);
```

```
break;
}
}

match vacant_box {
Some(box_nr) => ...,
None => ...,
}
```

Resolution: The Finding was fixed in commit 35f25c6.

I-2023-0039 - Unformatted Code - Info

Description: *cargo fmt* yields changes in 2 files total. Formatting the code is recommended for good Code Quality.

Assets:

- flashloan-pool/src/events.rs [https://github.com/gable-finance/gable/tree/main/src/backend/scrypto/flashloan-pool/]
- flashloan-pool/src/lib.rs [https://github.com/gable-finance/gable/tree/main/src/backend/scrypto/flashloan-pool/]

Status: Fixed

Recommendations

Remediation: Consider formatting the code using *rustfmt* or an equivalent.

Resolution: The Finding was fixed in commit 16dd2e3.

I-2023-0040 - The contract code is a single monolith file - Info

Description: In this commit, the *lib.rs*, main source file has 1009 lines. Splitting it into separate files/modules would increase its readability, hence its quality. Big chunks of logic can be extracted into separate functions, even if they are called once; just to make it easier to digest and reason about.

Assets:

- flashloan-pool/src/lib.rs [https://github.com/gable-finance/gable/tree/main/src/backend/scrypto/flashloan-pool/]

Status: Fixed

Recommendations

Remediation: Consider splitting large functions into smaller and/or moving the actual code of the contract's methods into a separate file.

Resolution: The Finding was fixed in commit 0b13351.

I-2023-0041 - Suggestions for idiomatic code style - Info

Description: *cargo clippy* is a popular tool for catching common mistakes and improving the code. It reports some possibly useful code changes.

Assets:

- flashloan-pool/src/events.rs [https://github.com/gable-finance/gable/tree/main/src/backend/scrypto/flashloan-pool/]
- flashloan-pool/src/lib.rs [https://github.com/gable-finance/gable/tree/main/src/backend/scrypto/flashloan-pool/]

Status: Fixed

Recommendations

Remediation: Consider following its suggestions and/or using `cargo clippy --fix` to apply some of them automatically

Resolution: The Finding was fixed in commit 82f9ac1.

I-2023-0042 - Former name is mentioned - Info

Description: The *README.md* file as well as *Functional Requirements* mentions the old protocol name.

Status: Fixed

Recommendations

Remediation: Consider replacing it with the relevant name.

Resolution: The Finding was fixed in commit 794794e.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/gable-finance/gable/tree/main/src/backend/scrypto/flashloan-pool/
Commit	afeb0343533798020630fcf45432abce7580b7e8
Whitepaper	Provided
Requirements	Provided
Technical Requirements	Provided

Contracts in Scope

flashloan-pool/src/events.rs

flashloan-pool/src/lib.rs