HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: The Sweat Foundation Ltd.
Date: 16 October, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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 The Sweat Foundation Ltd.
Approved By	Luciano Ciattaglia | Director of Services at Hacken OÜ
Lead Auditor	Noah Jelich | SC Lead Auditor at Hacken OÜ
Type	GameFi, Fungible Token
Platform	Near
Language	Rust
Methodology	Link
Website	https://sweateconomy.com/
Changelog	12.09.2023 - Initial Review 16.10.2023 - Second Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of Contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by The Sweat Foundation Ltd. (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

Sweat Economy is a system that allows users to earn $SWEAT tokens by walking using the following contracts:

- *sweat* - manages $SWEAT tokens.
- *sweat_jar* - allows staking $SWEAT for some interest.

The scope of this audit is the *sweat_jar* contract. Users can subscribe for a variety of Products. Subscription process is out-of-scope of the audit. Different Products have different terms like APY, lockup period, ability to restake or topup the stake. Based on the types of Products the user has access to, they can create Jars and deposit(stake) their $SWEAT tokens into them to earn some interest, specified in its Product terms.

Users can:

- create a jar(premium jars require the owner of the product to provide a signature)
- get information about Products and its terms
- get information about total amount staked, total interest accrued
- claim accrued interest at any moment
- restake $SWEAT in the Jar if its Product allows it and only after the lockup period has passed
- withdraw their $SWEAT after the lockup period(for Fixed Jars) or at any moment but with a fee(for Flexible Jars)

Privileged roles

Manager:

- Register a new product
- Disable an active product
- Update the public key for the specified product
- Upgrade or downgrade APY of any premium jar to predefined values

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Security Score

As a result of the audit, the code contains **2** medium and **4** low severity issues. The security score is **8** out of **10**.

All found issues are displayed in the Findings section of the report.

Summary

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
12 Sep 2023	2	4	3	1
16 Oct 2023	4	2	0	0

Risks

- Unless the smart contract is deployed with the `--final` flag, it could be upgraded and its functionality may be changed.
- State corruption due to cross-contract calls is a possibility.
- The $SWEAT minting process is out of scope. The protection mechanism to prevent the creation of multiple accounts, in order to claim more tokens by sending fake sensor values to the application, is **unknown**.
- The interest the users are eligible for staking is sent to the contract by the owners of the protocol **manually**.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

Critical

C01. Double Public Key Signing Function Oracle Attack On ed25519-dalek

Impact	High
Likelihood	High

Versions of ed25519-dalek prior to v2.0 model private and public keys as separate types, which can be assembled into a Keypair, and also provide APIs for serializing and deserializing 64-byte private/public keypairs.

Such APIs and serializations are inherently unsafe as the public key is one of the inputs used in the deterministic computation of the S part of the signature, but not in the R value. An adversary could somehow use the signing function as an oracle that allows arbitrary public keys as input can obtain two signatures for the same message sharing the same R and only differ on the S part.

This enables private key extraction attacks.

Paths: contract/src/product/model.rs, contract/src/jar/model.rs, contract/src/lib.rs, contract/src/ft_receiver.rs

Recommendation: Update to version >= 2.

Found in: ea17a63

Status: Fixed (Revised commit: a3f27ab)

High

H01. Requirements Violation

Impact	Medium
Likelihood	High

The *migrate_jars()* function in the system lacks adequate access control and signature verification mechanisms. Although the system generally mandates signature verification for premium products, the *ft_on_transfer()* function currently permits any entity to call *migrate_jars()* without undergoing any checks or validations. This oversight means that a malicious actor could potentially create jars for signature-required products without the signature verifications.

Paths: ./contract/src/ft_receiver.rs: ft_on_transfer()

./contract/src/migration/api.rs: migrate_jars()

Recommendation: Either implement a signature verification mechanism or restrict access only to the authorized actors.

Found in: ea17a63

Status: Fixed (Revised commit: a3f27ab)

Resolution: Access was restricted to contract managers

H02. Access Control Violation

Impact	Medium
Likelihood	High

The `*ft_on_transfer()*` function allows any actor to invoke the `*migrate_jars()*` function without any restrictions or validations. The `*migrate_jars()*` function is designed to establish a jar for a specified account with user-defined parameters. Among these parameters is the `*created_at*` variable, which plays a pivotal role in calculating interest for users.

A malevolent actor can exploit this oversight by invoking the function prior to the product's maturity date, intentionally setting the `*created_at*` parameter to zero. As the interest calculation is dependent on the `*created_at*` variable, this would allow the malicious user to artificially inflate their maturity time, enabling them to claim an unjustifiably high amount of $SWEAT tokens.

Paths: ./contract/src/ft_receiver.rs: ft_on_transfer()

./contract/src/migration/api.rs: migrate_jars()

Recommendation: The comment for the migrate function states, *"Reserved for internal service use; will be removed shortly after release."* If it is for internal use, restrict access to only authorized accounts.

Found in: ea17a63

Status: Fixed (Revised commit: a3f27ab)

Resolution: The `*migrate_jars()*` is now reserved for the owner. However, this functionality is temporary and as such, should be limited in time. See L03

H03. Inefficient Gas Model

Impact	Medium
Likelihood	High

The owner of the contract is paying for storage of the contract, so it is in their interest to keep the data usage as low as possible. The number of `*jars*` in the contract can grow infinitely and it never shrinks, even though some data in it is no longer useful.

Path: contract/src/lib.rs: Contract

Recommendation: Turn this structure from vector into map. Keep the nonce of the last jar so that every new jar has a unique nonce. Map this nonce to the actual jar. Remove jars after use.

Found in: ea17a63

Status: Fixed (Revised commit: a3f27ab)

Medium

M01. Missing Check

Impact	Medium
Likelihood	Medium

The *top_up()* function does not incorporate checks to determine if the associated product is disabled or if the jar's state is set to "Closed." As a result, users can still add funds (top-up) even when the product is not active or when the jar has been closed, leading to inconsistencies in the contract's behavior.

Path: contract/src/jar/model.rs: top_up()

Recommendation: Implement the necessary checks.

Found in: ea17a63

Status: Fixed (Revised commit: a3f27ab)

M02. Missing Annotation

Impact	Medium
Likelihood	Medium

The Near documentation stipulates that functions involving token transfers should be marked with the *payable* annotation. Functions such as *set_enabled()* and *set_public_key()* utilize the *assert_one_yocto()* function to facilitate token transfers, but they are not annotated as payable. This omission could lead to unexpected behavior or failures when these functions are invoked with $SWEAT token transfers.

Path: contract/src/product/api.rs: set_enabled(), set_public_key()

Recommendation: Implement the *payable* annotation to the specified functions.

Found in: ea17a63

Status: Fixed (Revised commit: a3f27ab)

M03. Undocumented Behavior

Impact	Medium
Likelihood	Medium

The contract charges a fee on withdrawal, but it is not mentioned in the public documents.

Paths: contract/src/ft_interface.rs: transfer(),

contract/src/withdraw/api.rs: withdraw(), transfer_withdraw()

Recommendation: Update public-facing documents, to clearly highlight the fee structure.

Found in: ea17a63

Status: Mitigated (Revised commit: a3f27ab)

Resolution: The documentation was updated with fee details

M04. Reliance On Off-Chain Data

Impact	Medium
Likelihood	Medium

In order to create a premium jar, the user has to first acquire a signature from the owner of the contract to prove his capability to create such a jar.

Path: contract/src/jar/model.rs: top_up()

Recommendation: Devise an on-chain access control for users and/or document it.

Found in: ea17a63

Status: Mitigated (Revised commit: a3f27ab)

Resolution: Documentation was updated with details of the migration procedure.

M05. Data Inconsistency Due to Edge Case

Impact	High
Likelihood	Low

There is an edge case in the claiming process when it is possible to get jar(s) stuck in a locked state. Such a jar would become undeletable and it would not be possible to claim interest from it. Withdrawing the principle would still be possible.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

This can happen because in *claim_jars()*, the jars get locked regardless of whether there is interest to claim from them or not. Normally, they get unlocked in a callback, but if there is no interest to claim yet at all, the callback will not be called and they get stuck in a locked state.

Path: ./contract/src/claim/api.rs: claim_jars()

Recommendation: Fortunately, this is easy to fix. Only lock the jars that are providing some interest.

```
for jar in unlocked_jars {
    let product = self.get_product(&jar.product_id);
    let available_interest = jar.get_interest(product, now);
    let interest_to_claim = amount.map_or(available_interest, |amount| {
        cmp::min(available_interest, amount.0 - total_interest_to_claim)
    });

    if interest_to_claim == 0 {
        continue;
    }

    jar.lock();
    ...
```

Found in: a3f27ab

Status: New

M06. State Corruption Due to Edge Case

Impact	High
Likelihood	Low

The *withdraw()* function does not check if the *jar.is_pending_withdraw*, meaning the cross-contract call is in progress for this jar. This can lead to unexpected behavior.

Path: ./contract/src/withdraw/api.rs: withdraw()

Recommendation: Perform a sanity check for *jar.is_pending_withdraw*.

Found in: a3f27ab

Status: New

Low

L01. Variable Is Not Limited

Impact	Low

Likelihood	Low

The contract does not validate the fee value provided, allowing it to be set equal to the withdrawal amount. Without proper checks and restrictions, this can lead to scenarios where users are unexpectedly charged high fees that match their withdrawal amount.

Path: contract/src/withdraw/api.rs : get_fee()

Recommendation: Provide conscious limits for stored configuration values.

Found in: ea17a63

Status: Reported (Revised commit: a3f27ab)

L02. Value Rounding

Impact	Low
Likelihood	Low

The interest is calculated using minutes. Compared to seconds or milliseconds, it reduces the precision of the calculation.

Path: contract/src/jar/model.rs: get_interest()

Recommendation: Perform calculations with better precision and/or document it.

Found in: ea17a63

Status: Fixed (Revised commit: a3f27ab)

L03. Jars Migration Functionality Is Not Limited in Time

Impact	Low
Likelihood	Low

Jars migration from the older version of the system is a scheduled and finite process. This functionality is designed to be temporary. It should have time limits - migration stage start (available at deployment) and stage end (after the timelock).

Path: ./contract/src/ft_receiver.rs: ft_on_transfer()

./contract/src/migration/api.rs: migrate_jars()

Recommendation: Implement a timelock mechanism, as it will be removed shortly after release. For instance, if the function is to be removed three days after release, add a timestamp and revert the function three days later if someone attempts to execute it. This can serve as an added safety measure to ensure that even if the function is not removed promptly, it becomes unusable after a certain period.

Found in: a3f27ab

Status: New

L04. Redundant Allocations

Impact	Low
Likelihood	Medium

Copying data wastes compute power and increases Gas fees.

- There are functions that unnecessarily allocate jars for the purpose of mutating fields of existing jars.

  Path: ./contract/src/jar/model.rs: Jar::{locked, unlocked, withdrawn}

  Recommendation: Consider removing these functions entirely. Mutate the fields of the target jar directly.

  Function *should_be_closed()* in *withdrawn()* should be used separately.

  In *withdraw()*, use *get_jar_mut_internal()* to get mutable reference, merge *do_transfer()* into *withdraw()* and use the reference to avoid copying the jar.

- Product keys are cloned and collected unnecessarily.

  Path: ./contract/src/migration/api.rs: migrate_jars()

  Recommendation: Consider the following:

```
// this line can be removed
let product_ids: Set<ProductId> =
self.products.keys().cloned().collect();
...
require!(
    // keys can be checked directly
    self.products.contains_key(&ce_fi_jar.product_id),
    format!("Product {} is not registered", ce_fi_jar.product_id),
);
```

- The jar is created just to get a value of 0 (*or_default()* case).

  Path: ./contract/src/jar/model.rs: verify()

  Recommendation: Consider getting an optional reference to the jar instead.

```
// this
```

```
let last_jar_id =
self.account_jars.entry(account_id.clone()).or_default().last_id;
// should just be this
let last_jar_id = self.account_jars.get(account_id).map(|jars|
jars.last_id);
// later you can remove the following line since last_jar_id is
already an Option
let last_jar_id = if last_jar_id == 0 { None } else {
Some(last_jar_id) };
```

- Redundant jar clone.

  Path: ./contract/src/migration/api.rs: migrate_jars()

  Recommendation: Consider pushing a jar after the event. This way you only need to clone *jar.account_id* and not the whole jar.

```
event_data.push(MigrationEventItem {
    original_id: ce_fi_jar.id,
    id: jar.id,
    account_id: jar.account_id.clone(),
});

account_jars.push(jar);
```

- String allocation can be omitted.

  Path: ./contract/src/ft_interface.rs: Promise::ft_transfer()

  Recommendation: Consider doing this:

```
let args = json!({
    "receiver_id": receiver_id,
    "amount": amount.to_string(),
    "memo": memo.unwrap_or_default(),
})
.as_str()
.map(|s| s.as_bytes().to_vec())
.unwrap_or_default();
```

Found in: a3f27ab

Status: New

L05. Inefficient Gas Model and Inconsistency

Impact	Low
Likelihood	Medium

At the moment, claiming interest is performed as follows:

1. The original jars are updated and their full old state is passed to callback. The jars get locked.
2. Tokens that make up interest are transferred to the user.
3. Callback function unlocks jars if token transfer was successful. If not, it rolls back the state of the jars.

Unlike this, during the withdrawal principle, the jars are only updated in the callback.

Secondly, to revert/apply updates, full instances of jars(sometimes all jars of the user) are passed to the callback. There is no need for this. Only the instructions for updating jars need to be passed forward. The jar only needs to be locked.

Path: ./contract/src/claim/api.rs: claim_jars()

Recommendation: Only collect and pass forward the information needed to revert/update jars in a callback. Possible fix:

```
#[derive(BorshDeserialize, BorshSerialize)]
struct JarClaimData {
    id: JarId,
    available_interest: u128,
    interest_to_claim: u128,
}

#[derive(BorshDeserialize, BorshSerialize)]
struct ClaimData {
    time: u64,
    account_id: AccountId,
    jars_data: Vec<JarClaimData>,
}

...
for jar in unlocked_jars {
    let product = self.get_product(&jar.product_id);
    let available_interest = jar.get_interest(product, now);
    let interest_to_claim = amount.map_or(available_interest, |amount| {
        cmp::min(available_interest, amount.0 - total_interest_to_claim)
    });

    if interest_to_claim == 0 {
        continue;
    }

    jar.lock();

    jars_data.push(JarClaimData {
        id: jar.id,
        available_interest,
        interest_to_claim,
    });
```

```
}
```

Then you can use it in the callback like so:

```
fn after_claim_internal(
    &mut self,
    claimed_amount: U128,
    ClaimData {
        time,
        account_id,
        jars_data,
    }: ClaimData,
    is_promise_success: bool,
) -> U128 {
    if is_promise_success {
        let mut event_data = vec![];

        for JarClaimData {
            id,
            available_interest,
            interest_to_claim,
        } in jars_data
        {
            event_data.push(ClaimEventItem {
                id,
                interest_to_claim: U128(interest_to_claim),
            });

            let jar = self.get_jar_mut_internal(&account_id, id);

            if jar.principal == 0 && available_interest ==
interest_to_claim {
                self.delete_jar(&account_id, id);
                continue;
            }

            jar.claim(available_interest, interest_to_claim,
time).unlock();

        }

        emit(EventKind::Claim(event_data));

        claimed_amount
    } else {
        for jar in jars_data {
            self.get_jar_mut_internal(&account_id, jar.id).unlock();
        }
```

```
        U128(0)
    }
}
```

Found in: a3f27ab

Status: New

Informational

I01. Redundant Configuration

`#[cfg_attr(not(target_arch = "wasm32"), derive(PartialEq))]` macro is supposed to ensure the code from `PartialEq` trait is only used in tests and is not included in the contract's code during deployment. However, it is unnecessary since, in this case, the compiler will strip unused code from the binary automatically.

Path: ./contract/src/*

Recommendation: Consider deriving the trait directly.

Found in: ea17a63

Status: Fixed (Revised commit: a3f27ab)

I02. Idiomatic Map Usage

The map is used in an ungraceful way. When a single element is being added, the whole map is cloned, modified, then saved back to storage.

Paths: ./contract/src/internal.rs: save_jar()

./contract/migration/api.rs: migrate_jars()

Recommendation: Consider the following usage:

`map.entry(key).or_default().insert(value);`

Found in: ea17a63

Status: Fixed (Revised commit: a3f27ab)

I03. Redundant Clones

Copying data wastes compute power and increases Gas fees.

Paths: ./contract/src/ft_receiver.rs: ft_on_transfer()

./contract/src/internal.rs: get_product(), save_jar()

./contract/src/product/model.rs: allows_top_up(), allows_restaking(),

./contract/src/product/api.rs: register_product()

Recommendation: Use a reference and only clone if and where a copy of data is required. Only clone a specific field of the structure.

Found in: ea17a63

Status: Fixed (Revised commit: a3f27ab)

I04. Code Readability

There is room for code quality improvement. Consider the following piece of code:

```
let fee = product.withdrawal_fee.clone()?;
let amount = match fee {
    WithdrawalFee::Fix(amount) => amount,
    WithdrawalFee::Percent(percent) => percent.mul(jar.principal),
};

Some(Fee {
    amount,
    beneficiary_id: self.fee_account_id.clone(),
})
```

Path: ./contract/src/withdraw/api.rs: get_fee()

Recommendation: Consider refactoring the function.

Found in: ea17a63

Status: Fixed (Revised commit: a3f27ab)

I05. Inefficient Gas Model

The *account_jars_with_ids()* function looks for a subset *ids* in a vector of jars, but does this in *O(jars.len() * ids.len())* time. This is not the problem if the counts are low; however, the more there are jars and the more there are *ids* to look for - the bigger the Gas fees.

Path: ./contract/src/internal.rs: account_jars_with_ids()
./contract/src/jar/api.rs: get_principal(), get_interest()

Recommendation: Consider using *HashMap* for constant time lookup. Also, the *ids* can be an owned value, a vector.

```
pub(crate) fn account_jars_with_ids(&self, account_id: &AccountId, ids:
Vec<JarIdView>) -> Vec<&Jar> {
    let mut result = vec![];

    // iterates once over jars and once over ids
    let jars: HashMap<U32, &Jar> = self
        .account_jars(account_id)
        .iter()
```

```
        .map(|jar| (U32(jar.id), jar))
        .collect();

    for id in ids {
        let &jar = jars
            .get(&id)
            .unwrap_or_else(|| env::panic_str(&format!("Jar with id: '{}'
doesn't exist", id.0)));
        result.push(jar);
    }

    result
}
```

A version of this function can also be used in *claim_jars()* as suggested in I09.

Found in: a3f27ab

Status: New

I06. Redundant Checks

Checking access to jars with *assert_ownership()* was useful in an older version. With current design, the ownership of the jar is guaranteed by default.

Path: ./contract/src/jar/api.rs: restake()

./contract/src/withdraw/api.rs: withdraw()

Recommendation: Consider removing this check.

Found in: a3f27ab

Status: New

I07. Inconsistent Interest Calculation

1. If *until_date <= base_date* then there is no need to calculate new *interest*.
2. *term_in_minutes* - variable name does not reflect its function. The calculation is actually done in milliseconds.

Path: ./contract/src/jar/model.rs: get_interest()

Recommendation: Consider the following:

1. Return *base_rate* early.
2. Rename it to something like *term_in_ms*.

Found in: a3f27ab

Status: New

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

I08. Redundant Operations

1. Since there is redundant copying of the jars, this function accepts a copy of one. There is no need for this since it will still be searched for in a collection.
2. There is no need to perform *swap_remove()* operation by hand.

Path: ./contract/src/jar/api.rs: restake()

Recommendation: Consider the following:

1. Accept user's id and their jar's id.
2. Use native *Vec::swap_remove()* function.

The whole function could look like this:

```
pub(crate) fn delete_jar(&mut self, account: &AccountId, jar_id: JarId) {
    match self.account_jars.get_mut(account) {
        None => env::panic_str(&format!("Account '{account}' doesn't
exist")),
        Some(jars) if jars.is_empty() => env::panic_str("Trying to delete
jar from empty account"),
        Some(jars) => {
            let jar_position = jars
                .iter()
                .position(|j| j.id == jar_id)
                .unwrap_or_else(|| env::panic_str(&format!("Jar with id
'{}' doesn't exist", jar_id)));
            jars.swap_remove(jar_position);
        }
    }
}
```

Found in: a3f27ab

Status: New

I09. Inefficient Search

Collecting *unlocked_jars* is done inefficiently (for large collections of jars).

Path: ./contract/src/claim/api.rs: claim_jars()

Recommendation: Consider using a version of the function you already have (I05) except to get mutable references to be able to lock the jars.

```
let unlocked_jars: Vec<&mut Jar> =
self.account_jars_with_ids_mut(&account_id, jar_ids)
    .into_iter()
    .filter(|jar| !jar.is_pending_withdraw)
    .collect();
```

Found in: a3f27ab

Status: New

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	**Critical**	**High**	**Medium**
Medium Likelihood	**High**	**Medium**	**Low**
Low Likelihood	**Medium**	**Low**	**Low**

Risk Levels

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 won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Appendix 2. Scope

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

Initial Review Scope

Repository	https://github.com/sweatco/sweat-jar
Commit	ea17a632de99ff78a33368401cdcdf2e74792896
Requirements	https://github.com/sweatco/sweat-jar/blob/main/README.md
Contracts	File: contract/src/assert.rs SHA3: 331e7cebb01f7e185d198b02852c941c5323688dc381ad84c7b485f3fde514bb File: contract/src/common.rs SHA3: 201592781295e5713ed18ddf4997586db1e3f9dae16b84759f8157eba01cc6d8 File: contract/src/event.rs SHA3: 0028078a44ca9b9dc5e8801b6f3f1375a13d8cf788f222d578af2fba8e5429dc File: contract/src/ft_interface.rs SHA3: d164dfa7480261e1d4b553681cf1f59be4fe8029487beba5257df339276f9d35 File: contract/src/ft_receiver.rs SHA3: 51be5ed4ce8f1b559f76edeb3d3ebd3ce59bb0038d0b00a9dca640f50fdf0c26 File: contract/src/internal.rs SHA3: f8b233d4f2e286585d298562eb3d632db60a227dbaf71294acf99016e95fc07f File: contract/src/lib.rs SHA3: cad3cd84ca6c5528963831fc52cbff301ce779985d096135e6a42c469c49c254 File: contract/src/claim/api.rs SHA3: da821dde1baeb6a536f565861944d7481dc4b0b6758e70cbc8e0d1915e65b5ca File: contract/src/claim/mod.rs SHA3: 79393c4deae75264739c390c66121e1b1f7f43b0c54a335176522a55b867280f File: contract/src/jar/api.rs SHA3: 428543b719c88c8c1fffe8e7d71c85ff5570fc5f250f12c32d5eac4d4ef24e0f File: contract/src/jar/mod.rs SHA3: e7731cec1cc05f8f04b3f46fc199b47193b17851944bcbd1970e6cad911b1601 File: contract/src/jar/model.rs SHA3: a2abdbb9ebfc2b4401cdca8ae92852c74b627e15a251afff8bbd0e27a246d71f File: contract/src/jar/view.rs SHA3: 4bb126053ab86b1cea4e98a43ba021078ac603bddc547b0e5f8f21c6967e9c1a File: contract/src/migration/api.rs SHA3: 828aab67fbcc4acdb4936391d1666537962af02b3005aaa273d84ce6982278ec File: contract/src/migration/mod.rs SHA3: 00a968a132480ab09b8d22f9a84a24cb8e7e78914788066c2310b4b6127aaef1 File: contract/src/migration/model.rs SHA3: 4303cfd6f3775de04a0b0d3ce44e162925c9ee0b0d01a26b69bd795fea7a708b File: contract/src/penalty/api.rs

	SHA3: 91df74779c1c9a1a16a58d673357922348fcf65418ed1a37d3da0bf67476aab9 File: contract/src/penalty/mod.rs SHA3: 78901a0d8657a53e24217cf108c8d5657ca6d7a89d02adaa2ce1d32cb5f9df49 File: contract/src/product/api.rs SHA3: 89d2b225b7bd8b4b4b817aa0cd54e7663466cfa24ff30d604ca8218ad884338c File: contract/src/product/command.rs SHA3: afe2602b233739dbda1190138f8352afdc12a14cf78edbd39096f09073d495e9 File: contract/src/product/mod.rs SHA3: 111311d32e1035c6d6342c14f640642a899ad158cff694fabf5aec32b59cc6e2 File: contract/src/product/model.rs SHA3: 07fe258a991014c4d522dcc9774ad21c1c7bea31df27ce2a3e7df09688a90c5c File: contract/src/product/view.rs SHA3: 5e9609e9a67a6264d19402730111c3fd6d1a9fda1867ddfafde34eb43c9497cf File: contract/src/withdraw/api.rs SHA3: ad57108cbae453ba28112936a1878f8ee58c84dea105129d81668006d83368e2 File: contract/src/withdraw/mod.rs SHA3: 08730d2756bd3e43db3dbd988e3067dfb89c54f4bd9f781811f53a25c05968b6 File: contract/src/withdraw/view.rs SHA3: f3fa67925c112893b333f34eaa4f4109410c729f72e9eec6dd17112322aa3131

Second Review Scope

Repository	https://github.com/sweatco/sweat-jar
Commit	a3f27ab6b92e50e0230cecefbb63e3e9c2d1a539
Requirements	https://github.com/sweatco/sweat-jar/blob/a3f27ab6b92e50e0230cecefbb63e3e9c2d1a539/README.md
Contracts	File: ./claim/mod.rs SHA3: d80f2f57d601225d993b08efba86d75bb533f8df3b63dd4895c8a4659c1d5d7d File: ./claim/api.rs SHA3: fc50a2eec2492835a5ba0ca5155494bf76c7da0254c8222061e4278375495909 File: ./common/udecimal.rs SHA3: fedc8767f64d3369657c74a06b2ccaf4a1291e5aea2f401f5c7fb0ccb34ccbe8 File: ./common/mod.rs SHA3: 631c77e83d2f0eed42bcbd2301eac59580526ef00529bbd93a98e9a534eb1d22 File: ./common/u32.rs SHA3: 8fc9e64bd793a730b5a167d7a5eb5e8cfcdcdb2f7bf14d5bf3d3dbd3aa0e0178 File: ./jar/mod.rs SHA3: c14734a28aa7b79af57e78b8647d3fb5e6c2da3b084a632795fd5030512e8775 File: ./jar/api.rs SHA3: e5534114c015beb0ff7f5dc84ef000de698de3e00ac2ef401e0784dc7817567d File: ./jar/model.rs SHA3: dca17cba5d8bc667a307f1feb8c5d1fef0e838b98d63d0f33dd2014fc21a6e6d File: ./jar/view.rs

SHA3: be7708c37411f8eddf75114d156f124e6b3aa353a2b858f07821cb440d55a989

File: ./migration/mod.rs
SHA3: 00a968a132480ab09b8d22f9a84a24cb8e7e78914788066c2310b4b6127aaef1

File: ./migration/model.rs
SHA3: e0fe54fa0a1a59910163f53b0f05407d4bfa9850b6eb948d49aa8ae19f97b087

File: ./migration/api.rs
SHA3: 416dbd7294d568d3053602dac469e0b66ec08e5e8e8ab87eb666b625d0ab1851

File: ./penalty/mod.rs
SHA3: 78901a0d8657a53e24217cf108c8d5657ca6d7a89d02adaa2ce1d32cb5f9df49

File: ./penalty/api.rs
SHA3: e060960fb21a707f8bafc908730e11113980421901ef763904c2d7b027dcb0d4

File: ./product/mod.rs
SHA3: 99b0fa10c32235d36b3abf9db1e8fda3d22b5b63c16509d50f2e71442edffaa3

File: ./product/api.rs
SHA3: 17d560cbd039e9745ae712797d2ab16a66b71e95173749b720467639683ba0ee

File: ./product/command.rs
SHA3: 7279f2ea1b2804e520fea7a304b14dc2b2db565f9abbe7d9eecf7fa82be136c5

File: ./product/model.rs
SHA3: d76d59c89365ca46f3d8c2e70ce92c92684363124275514f49a8a4cd508d6d21

File: ./product/view.rs
SHA3: e366b23370eb88fc6d8407a408fd1841f1c5d7f24d2546ca83d1d8cfbe7469a1

File: ./withdraw/api.rs
SHA3: 804b179e569e37b1a9b0d6cef86f1acd21754107754d6fcba643ef1c63bbd598

File: ./withdraw/mod.rs
SHA3: 80777cdcee9abb14da30aebb5206a2c7cbf91b0ad49bb91e9ee2ff99451c9945

File: ./withdraw/view.rs
SHA3: 1133cc410e7ac8e72fa31faddbe7d61e210d735963938f2889ca428bedaf74c3

File: ./assert.rs
SHA3: 839e3a08a3ba07f3a0013598907dc678d96da8221261222a6d0aa78a03d9aac3

File: ./event.rs
SHA3: f8ad2a7831d337d85ec1d5596db6154e01c8ee9045b20179d01ed956f410de4f

File: ./ft_interface.rs
SHA3: 908b57bdee4ab945abc1e4addba080fea99431d86d58041705dde2b29c1e8127

File: ./ft_receiver.rs
SHA3: e2328a42d2472c117a888ec49f3ba5182e21315f0bd38f1e698ab08d45d6d18a

File: ./internal.rs
SHA3: fed20403bfa0af5795552f0ff15c3f16f947f4797bf82890334553416e6f064a

File: ./lib.rs
SHA3: 8779ad5459c0d8fa8b8c848ad180f1b29640c92fe0b8881775536bfb82147772