Guvenkaya® The Bedrock of Security

The Sweat Foundation Ltd

Sweat Jars Migration And Refactor Smart Contract Review

Lead Security Engineer: Timur Guvenkaya
Date of Engagement: 7th May 2025 - 12th May 2025
Visit: www.guvenkaya.co

Contents

About Us

Guvenkaya is a security research firm specializing in Rust security, Web3 security of Rust-based protocols, and Web2 security. With our expertise, we provide both security auditing services and custom security solutions

About The Sweat Foundation

The Sweat Foundation is an organization behind Sweat Economy, an innovative project at the intersection of fitness and crypto. It motivates users to stay active by converting their steps into SWEAT Token. This approach promotes health and fitness and works as an entry point to crypto for many users.

Audit Results

Guvenkaya conducted a security assessment of the Sweat migration and refactor changes inside the Sweat Jar smart contract. During this engagement, 2 findings were reported. One is Medium severity and another is Ingormational severity. The Sweat Foundation team has fixed the medium issue and acknowledged the informational one.

Project Scope

Sweat Jar Contract

File name	Link
V2 Migration	https://github.com/sweatco/sweat-jar/blob/5892a645dc58e5e1c6b186568af9313bfe0de844/contract/src/migration/v2.rs
Migration	https://github.com/sweatco/sweat-jar/tree/77ad0664f6efc989013b5ea9be16ab63aac9488e/contract/src/migration
Model	https://github.com/sweatco/sweat-jar/tree/77ad0664f6efc989013b5ea9be16ab63aac9488e/model/src
FT Receiver	https://github.com/sweatco/sweat-jar/blob/77ad0664f6efc989013b5ea9be16ab63aac9488e/contract/src/feature/ft_receiver.rs
Withdraw All	https://github.com/sweatco/sweat-jar/blob/77ad0664f6efc989013b5ea9be16ab63aac9488e/contract/src/feature/withdraw/api.rs#L120
Restake	https://github.com/sweatco/sweat-jar/tree/77ad0664f6efc989013b5ea9be16ab63aac9488e/contract/src/feature/restake

Out of Scope

The audit will include, but is not limited to, reviewing the code for security vulnerabilities, coding practices, and architecture. The audit does not include a review of the dependencies.

Scope only includes:

- New Migration Functionality: Migration code in new contract and old contract
- Restaking changes: Addition of ticketing system to restake and restake_all
- Withdrawal Changes: withdraw_all accepting optional product arguments
- Codebase refactor: Moving non-NEAR related code to the models folder. High level review

Timeline

Start of the audit	Draft report	Final report
7th May 2025	12th May 2025	12th May 2025

Methodology

- RESEARCH INTO PROJECT ARCHITECTURE
- PREPARING ATTACK VECTORS
- SETTING UP AN ENVIRONMENT
- MANUAL CODE REVIEW OF THE CODE
- ASSESSMENT OF RUST SECURITY ISSUES
- ASSESSMENT OF NEAR SECURITY ISSUES
- ASSESSMENT OF ARITHMETIC ISSUES
- BUSINESS LOGIC VULNERABILITY ASSESSMENT
- ONCHAIN TESTING USING NEAR WORKSPACES
- BEST PRACTICES AND CODE QUALITY
- CHECKING FOR CODE REFACTORING/SIMPLIFICATION POSSIBILITIES
- ARCHITECTURE IMPROVEMENT SUGGESTIONS
- PREPARING POCS AND/OR TESTS FOR EACH CRITICAL/HIGH/MEDIUM ISSUES

Severity Breakdown

01. Likelihood Ratings

Likely: The vulnerability is easily discoverable and not overly complex to exploit.
Possible: The vulnerability presents some challenges either in discovery or in the complexity of the attack.
Rare: The vulnerability is either very difcult to discover or complex to exploit, or both.
This matrix provides a nuanced view, taking into account both the ease of discovering a vulnerability and the complexity involved in exploiting it.

02. Impact

Severe: Exploitation could result in critical loss or compromise, such as full system control, substantial financial loss, or severe reputational damage.
Moderate: Exploitation may lead to limited data loss, partial compromise, moderate financial impact, or noticeable degradation of services.
Negligible: Exploitation has minimal impact, such as minor data exposure without significant consequences or slight inconvenience without substantial disruption.

03. Severity Ratings

Critical: Assigned to vulnerabilities with severe impact and a likely likelihood of exploitation.
High: For vulnerabilities with either severe impact but only a possible likelihood, or moderate impact with a likely likelihood.
Medium: Used for vulnerabilities with severe impact but a rare likelihood, moderate impact with a possible likelihood, or negligible impact with a likely likelihood.
Low: For vulnerabilities with moderate impact and rare likelihood, or negligible impact with a possible likelihood.
Informational: The lowest severity rating, typically for vulnerabilities with negligible impact and a rare likelihood of exploitation.

CRITICAL **HIGH** **MEDIUM** Low Informational

Likelihood Matrix:

Attack Complexity \ Discovery Ease	Obvious	Concealed	Hidden
Complex	Possible	Rare	Rare
Moderate	Likely	Possible	Rare
Straightforward	Likely	Possible	Possible

Likelihood/Impact Matrix:

Likelihood \ Impact	Severe	Moderate	Negligible
Likely	**CRITICAL**	**HIGH**	**MEDIUM**
Possible	**HIGH**	**MEDIUM**	Low
Rare	**MEDIUM**	Low	Informational

Findings Summary

01. Remediation Complexity: This measures how difcult it is to fx the vulnerability once it has been identifed.

Simple: Patches or fixes are readily available and easily implemented.
Moderate: Requires some time and resources to remediate, but well within the capabilities of most organizations.
Difficult: Remediation requires significant resources, specialized skills, or substantial changes to systems or architecture.

02. Status: This measures how difcult it is to fx the vulnerability once it has been identifed.

Not Fixed: Indicates that the vulnerability has been identifed but no remedial action has been taken yet. This status is crucial for newly discovered vulnerabilities or those awaiting prioritization.
Fixed: This status is applied when the vulnerability has been successfully remediated. It implies that appropriate measures (like patching, confguration changes, or architectural modifcations) have been implemented to resolve the issue.
Acknowledged: This status is used for vulnerabilities that have been recognized, but for various reasons (such as risk acceptance, cost, or other business decisions), have not been fxed. It indicates that the risk posed by the vulnerability is known and has been consciously accepted.

Finding	Impact	Likelihood	Severity	Remediation Complexity	Remediation Status
GUV-1: Possible Double Claim Through Race Condition	Severe	Rare	**MEDIUM**	Simple	Fixed
GUV-2: Cross Function Ticket Reuse	Negligible	Rare	Informational	Simple	Fixed

Findings Details

GUV-1 Possible Double Claim Through Race Condition - Medium

We observed that it is not verified whether a user is in migrating state through **assert_account_is_not_migrating** in any of the state changing functions like claim/withdraw/restake in old contract. Since the account is deleted only inside of the **after_account_transferred**, there is a time for a malicious actor to call **claim** to claim accumulated amount and then after the account is deleted, call claim again on a new migrated smart contract to claim twice.

migration:contract/src/migration/v2.rs

```
pub fn after_account_transferred(&mut self, account_id: AccountId) ->
PromiseOrValue<(AccountId, bool)> {
        let is_success = is_promise_success();

        if is_success {
         self.clear_account(&account_id);
           emit(EventKind::JarsMerge(account_id.clone()));
        }
        ...
```

PROPOSED SOLUTION

Consider asserting the **assert_account_is_not_migrating** in all state mutating functions.

REMEDIATION - FIXED

The Sweat Foundation team has fixed the issue by adding the **assert_account_is_not_migrating** in all state mutating functions in this commit: **89755f5cfbac88764131560ec1d0085464f58cc9**

GUV-2 Cross Function Ticket Reuse - Informational

We observed that it is possible to generate a ticket with a signature for jar creation but supply it to the **restake** and **restake_all** functions. Depending on logic of both functions, it can introduce security issues in the future and in general it increases the attack surface.

migration:contract/src/feature/restake/api/api.rs

```
    fn restake(
            &mut self,
           from: ProductId,
           ticket: DepositTicket,
           signature: Option<Base64VecU8>,
           amount: Option<U128>,
         ) -> PromiseOrValue<()> {
```

migration:contract/src/migration/v2.rs

```
    pub(crate) fn create_jar(
            &mut self,
           account_id: AccountId,
           ticket: JarTicket,
           amount: U128,
           signature: Option<Base64VecU8>,
         ) -> JarView
```

PROPOSED SOLUTION

Consider disallowing cross-function ticket reuse.

REMEDIATION - FIXED

The Sweat Foundation team has fixed the issue by introducing purpose in deposit message in this commit and update on the backend: **fd72e8cb72af66eb254d90c173387bf8236e61f3**