Guvenkaya® The Bedrock of Security

The Sweat Foundation Ltd

Burn Model NEAR Rust Smart Contract Security Assessment

Lead Security Engineer: Timur Guvenkaya
Date of Engagement: 22nd April 2024 - 26th April 2024
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 Economy new burn model from 22nd April 2024 to 26th April 2024. During this engagement, a total of 3 findings were reported. All of the findings are low severity. All issues are fixed by the Sweat Foundation team.

Project Scope

Commit Hash:

8fcaa54079dfa69cc6f57ae8d2644de4001ed427

File name	Link
Migration	src/migration/migration.rs
Record API	src/record/api.rs
Claim API	claim/api.rs
Account Record	https://github.com/sweatco/sweat-claim/blob/8fcaa54079dfa69cc6f57ae8d2644de4001ed427/model/src/account_record.rs#L159-L170

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. Only the changes to the burn model are in scope.

Timeline

Start of the audit	Final report
22nd April 2024	28th April 2024

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: The vulnerability is easily discoverable and not overly complex to exploit.
Moderate: The vulnerability presents some challenges either in discovery or in the complexity of the attack.
Negligible: The vulnerability is either very difcult to discover or complex to exploit, or both.

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 difficult it is to fix the vulnerability once it has been identified.

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 difficult it is to fix the vulnerability once it has been identified.

Not Fixed: Indicates that the vulnerability has been identified 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, configuration changes, or architectural modifications) 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 fixed. 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: Error Prone Burn Rate Calculation	Moderate	Rare	Low	Simple	Fixed
GUV-2: Error Prone Getting Account API	Moderate	Rare	Low	Simple	Fixed
GUV-3: Lack of Prepaid Gas Checking Before Cross Contract Calls	Moderate	Rare	Low	Simple	Fixed

Findings Details

GUV-1 Error Prone Burn Rate Calculation - Low

We observed that **get_burn_rate** uses floating point arithmetic in smart contracts, and then casts the result from a floating point number to a fixed point one. This approach is prone to errors as it could potentially cause truncation errors.

model:get_burn_rate:model/src/lib.rs

```
    #[allow(
        clippy::cast_possible_truncation,
        clippy::cast_precision_loss,
        clippy::cast_sign_loss
    )]
    pub fn get_burn_rate(balance: TokensAmount, burn_period: Duration) ->
TokensAmount {
        (balance as f64 / f64::from(burn_period)).ceil() as u128
    }
```

PROPOSED SOLUTION

We propose using **div_ceil** which handles the division and the ceiling of the result, without utilizing floating point arithmetic.

REMEDIATION - FIXED

The Sweat Foundation team has fixed the issue by replacing the calculation with **div_ceil** in this commit: **18a8f9d7c064f6fe89960eb7e07e1b0333faabff**

GUV-2 Error Prone Getting Account API - Low

We noticed that the **get_account_mut** method not only gets the mutable reference to the account, but also inserts the account if it does not exist. This could lead to unexpected behavior and potential security vulnerabilities if this method is used without checking the implementation.

common:get_account_mut:contract/src/common/mod.rs

```
            fn get_account_mut(&mut self, account_id: &AccountId) -> &mut AccountRecord {
             if !self.contains_key(account_id) {
               self.insert(account_id.clone(),
      AccountRecordVersioned::new(now_seconds()));
              }

              let AccountRecordVersioned::V1(account) =
      self.get_mut(account_id).expect("Account not found");
              account
            }
```

PROPOSED SOLUTION

We propose removing an account creation functionality from **get_account_mut** and creating a separate function called **get_or_insert_account_mut** which will use **get_account_mut**.

REMEDIATION - FIXED

The Sweat Foundation team has fixed the issue by implementing suggested fix in these commits:
417d42ea895aaf9193501b3b33ccc8b8a91e35b1
a0a9e4057a833b2d60c12ee7038a556dc9e8edde

GUV-3 Lack of Prepaid Gas Checking Before Cross Contract Calls - Low

We noticed that in both claim and burn methods, the contract does not check whether there is enough prepaid gas for cross-contract calls and callbacks. This could lead to the "Exceeded the prepaid gas" errors if not enough gas was attached

PROPOSED SOLUTION

We propose checking whether **env::prepaid_gas()** is enough for the cross-contract call and callback before making a call.

REMEDIATION - FIXED

The Sweat Foundation team has fixed the issue by taking a difference between env::prepaid_gas and env::used_gas and asserting it in both claim and burn methods before cross-contract calls in this commit: **06c59cdb1476d173de96a68ff82c00d4bf17e8bf**