# FomoWell Security Audit Report

Prepared by VelintSec
Version 1.3

**Lead Auditors**
Max Ma
Vincent

Aug 26th ,2025

## Table of contents

## 1 About VelintSec

VelintSec is a full-service security service company based on the blockchain and Web3 ecosystem, focusing on the ICP ecosystem, focusing on providing security audit, protection and consulting services for smart contracts, blockchain projects and centralized systems.Our mission is to establish and maintain the highest security standards in the blockchain industry through rigorous auditing, continuous innovation, and dedicated support.. Learn more about us at velintsec.com.

## 2 Disclaimer

The VelintSec team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the The Internet Computer (ICP) implementation of the contracts.

## 3 Risk Classification

| | **Impact: High** | **Impact: Medium** | **Impact: Low** |
|---|---|---|---|
| **Likelihood: High** | Critical | High | Medium |
| **Likelihood: Medium** | High | Medium | Low |
| **Likelihood: Low** | Medium | Low | Low |

## 4 Protocol Summary

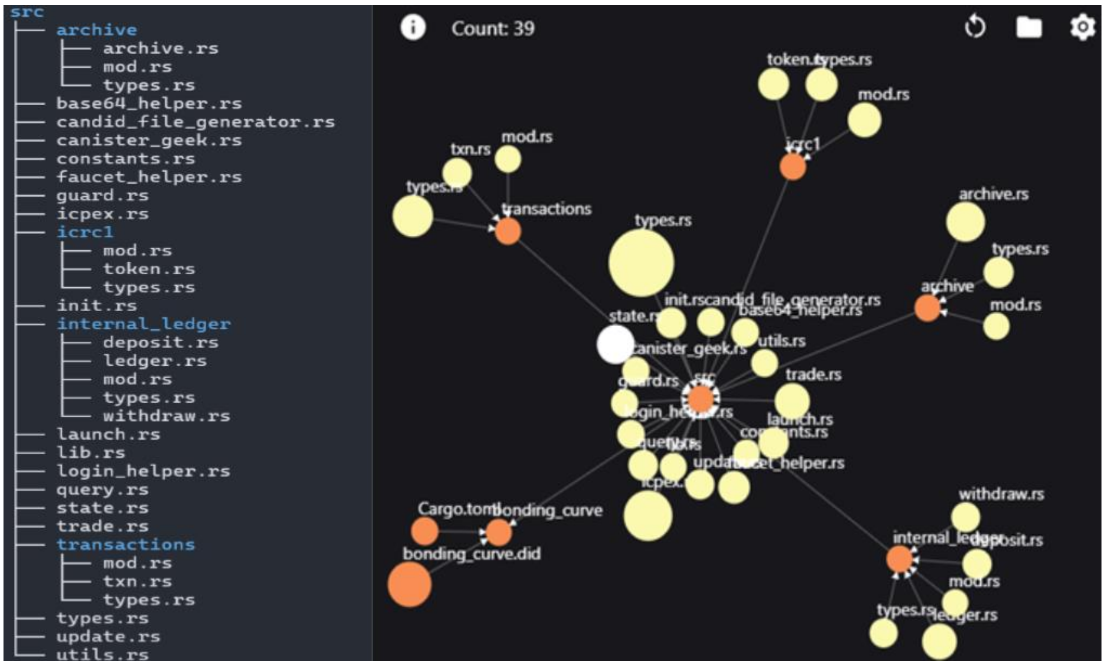
FomoWell is a decentralized meme coin launchpad built on the Internet Computer (IC), designed to provide a creative and scalable ecosystem for creators, investors, and communities. By leveraging IC's high-performance blockchain technology and fostering deep collaboration with community projects, FomoWell simplifies the process of meme coin issuance and provides new momentum for decentralized economies and community-driven ecosystems.

FomoWell stands out as more than just a tool for launching meme coins. It is a powerful platform that combines blockchain transparency with community empowerment, bringing real value and long-term vitality to the meme culture.

FomoWell has the following key technical features:

- Full-Stack Development on IC
- Security and Trading System
- High-Precision Data Analysis and Visualization
- Innovative Cross-Module Integration
- Community Support and Customization

The function invocation relationship of the fomowell project code is as follows:



```
src
├── archive
│   ├── archive.rs
│   ├── mod.rs
│   └── types.rs
├── base64_helper.rs
├── candid_file_generator.rs
├── canister_geek.rs
├── constants.rs
├── faucet_helper.rs
├── guard.rs
├── icpex.rs
├── icrc1
│   ├── mod.rs
│   ├── token.rs
│   └── types.rs
├── init.rs
├── internal_ledger
│   ├── deposit.rs
│   ├── ledger.rs
│   ├── mod.rs
│   ├── types.rs
│   └── withdraw.rs
├── launch.rs
├── lib.rs
├── login_helper.rs
├── query.rs
├── state.rs
├── trade.rs
├── transactions
│   ├── mod.rs
│   ├── txn.rs
│   └── types.rs
├── types.rs
├── update.rs
└── utils.rs
```

## 5 Audit Scope

The following contracts were included in the scope for this audit:

| Path | Files | Code | Comment | Blank | Total |
|---|---|---|---|---|---|
| src\bonding_curve | 31 | 3,615 | 91 | 420 | 4,126 |
| src\bonding_curve\src | 31 | 3,615 | 91 | 420 | 4,126 |
| src\bonding_curve\src (Files) | 17 | 2,230 | 79 | 264 | 2,573 |
| src\bonding_curve\src\archive | 3 | 315 | 5 | 50 | 370 |
| src\bonding_curve\src\icrc1 | 3 | 371 | 3 | 33 | 407 |
| src\bonding_curve\src\internal_ledger | 5 | 335 | 3 | 42 | 380 |
| src\bonding_curve\src\transactions | 3 | 364 | 1 | 31 | 396 |

## 6 Executive Summary

Over the course of 90 days, the VelintSec team conducted an audit on the Fomowell smart contracts provided by NovaICLabs. In this period, a total of 10 issues were found. The findings consist of 6 High risk,3 Medium risk,1 Low risk severity issues with the remainder being informational.

**Summary**

| | |
|---|---|
| Project Name | fomowell_btc_marketplace |
| Repository | fomowell_btc_marketplace_develop |
| Commit | 553d127c26bbc6594d785b4f7f131c5882c6ee9e |
| Audit Timeline | May 21st – Aug 26th, 2025 |
| Methods | Manual Review |

**Issues Found**

| | |
|---|---|
| **Critical Risk** | **0** |
| **High Risk** | **6** |
| **Medium Risk** | **3** |
| **Low Risk** | **1** |
| **Total Issues** | **10** |

**Summary of Findings**

| | | |
|---|---|---|
| **VLS_FMW_001** | [Hight]-- Potential Integer Overflow Vulnerability *./bonding_curve/src/swap/pool.rs* | **Fixed** |
| **VLS_FMW_002** | [Hight]-- Lack of Emergency Response Plan in case of Suspension of Trading or Withdrawal *./bonding_curve/src/* | **Fixed** |
| **VLS_FMW_003** | [Hight]-- Fees and LP benefits are Directly Credited to the Swap Pool *./bonding_curve/src/swap/pool.rs* | **Fixed** |
| **VLS_FMW_004** | [Hight]-- Unsafe Media Upload / Metadata Injection Vulnerability *./bonding_curve/src/types.rs* | **Fixed** |
| **VLS_FMW_005** | [Hight]-- Insecure URL Validation Vulnerability *./bonding_curve/src/types.rs* | **Fixed** |
| **VLS_FMW_006** | [Hight]-- Unsafe URL Validation Leading to Open Redirect & SSRF *./bonding_curve/src/types.rs* | **Fixed** |

| | | |
|---|---|---|
| **VLS_FMW_007** | [Hight]-- Stored XSS via Unsanitized Token Description<br>*./bonding_curve/src/types.rs* | **Fixed** |
| **VLS_FMW_008** | [Hight]-- Stored XSS via Unsanitized Twitter Handle Field<br>*./bonding_curve/src/types.rs* | **Fixed** |
| **VLS_FMW_009** | [Hight]-- Stored XSS via Unsanitized Telegram Handle Field<br>*./bonding_curve/src/types.rs* | **Fixed** |
| **VLS_FMW_010** | [Low]-- Insufficient error handling<br>*./bonding_curve/src/internal_ledger/ledger.rs* | **Fixed** |

# 7 Findings

## 7.1 High Risk

### 7.1.1 VLS_FMW_001: Potential Integer Overflow Vulnerability

**Description:**

The change_fee function splits the base BTC amount proportionally into a fee (fee), a burn amount (burn), and a remaining amount (btc - fee - burn). The fee and burn are calculated as btc * fee_ / 1_000_000u128 and btc * burn_ / 1_000_000u128, respectively. If the sum of the input parameters fee_ (fee ratio) and burn_ (burn ratio) exceeds 1_000_000u128 (i.e., the sum of the ratios exceeds 100%), the result of the fee + burn calculation will be greater than the original BTC amount. Because u128 in Solidity is an unsigned integer type, executing btc - fee - burn in this case will trigger an integer subtraction overflow. Negative results will wrap around to a very large positive number (for example, -100 becomes 2^128 - 100), causing the returned remaining amount to deviate significantly from business logic expectations.

```
// src/bonding_curve/src/swap/pool.rs
pub fn change_fee(btc: u128, fee_: u128, burn_: u128) -> (u128, u128, u128) {
    let fee = btc * fee_ / 1_000_000u128;
    let burn = btc * burn_ / 1_000_000u128;
    (btc - fee - burn, fee, burn)
}
```

**Impact:**

This vulnerability can lead to severe business logic failure and asset anomalies. On one hand, the remaining amount should represent the remaining assets after the distribution of fees and destruction, but it will display as an extremely large positive number (for example, if a user deposits 1000 units of assets, with a fee plus destruction ratio of 110%, theoretically there should be no remaining, but after overflow, it will show a remaining of 2^128 - 100 units), thus destroying the credibility of the contract data. On the other hand, attackers can exploit this vulnerability to construct malicious parameters, combine them with other permission defects to illegally transfer assets, or interfere with the subsequent logic of the contract (such as asset distribution, balance update) through abnormal remaining amounts, leading to unpredictable financial losses or system chaos.

**Recommended Mitigation:**

The core of the fix is to prevent the fee + burn from exceeding the BTC limit.

First, add parameter validation at the function entry point, enforcing that fee_ + burn_ ⩽ 1_000_000u128 (i.e., the sum of the proportions does not exceed 100%), thus preventing overflow conditions from being triggered at the source. An example code is require(fee_ + burn_ <= 1_000_000u128, "fee_ + burn_ exceeds 1,000,000").

Second, use the checked_sub method of a safe math library instead of direct subtraction to explicitly check for subtraction overflow and automatically roll back abnormal transactions, enhancing code robustness.

Additionally, optionally limit the maximum values of fee_ and burn_ (e.g., limit both to 1_000_000u128) to further mitigate potential risks caused by excessively large individual proportions.

NovaICLabs: Fixed at 4c78a53 and 91b00a6

VelintSec: Confirmed

### 7.1.2 VLS_FMW_002: Lack of Emergency Response Plan in case of Suspension of Trading or Withdrawal

**Description:**

Due to the decentralized nature of the DeFi system, the irreversibility of smart contracts, and the complexity of high-frequency trading scenarios, sudden suspensions caused by technical failures, security threats, and extreme market fluctuations may lead to liquidity interruptions, user asset freezes, market panic, and even systemic risks. Therefore, a rapid freezing and isolation mechanism is needed.

**Impact:**

If DeFi applications lack emergency response plans, technical vulnerabilities (such as smart contract errors) or security incidents (such as hacker attacks) may result in the inability to quickly freeze risks when trading/withdrawal functions are abnormal, triggering large-scale asset freezing and user panic runs; the lack of emergency support during extreme market fluctuations (such as liquidity crises) will aggravate chain liquidations and cause huge losses to user assets; at the same time, the inability to explain the situation to users in a timely manner will lead to a collapse of trust, resulting in user loss and even facing compliance investigations and penalties from regulators.

**Recommended Mitigation:**

A global pause function should be preset (such as a smart contract pause mechanism triggered by multi-signatures) to quickly freeze abnormal transactions; a hierarchical response should be implemented to handle technical failures (switching to a backup chain), security incidents (freezing the addresses involved and tracking funds), and market crises (injecting liquidity buffers); users should be notified of the reasons for the suspension and the recovery plan in real time via email to maintain transparency; and third-party audits should be conducted afterwards to review vulnerabilities, repair and update emergency procedures, and enhance system resilience.

NovaICLabs: Fixed at 4ee77f8

VelintSec: Confirmed

### 7.1.3 VLS_FMW_003: Fees and LP benefits are Directly Credited to the Swap Pool

**Description:**

If DeFi applications directly include transaction fees and LP returns in the Swap Pool (liquidity pool), the total amount and proportion of assets in the pool will be artificially manipulated. After the fees or reward tokens are injected, the original asset balance is broken (for example, token A is in excess and token B is scarce). At the same time, LP returns are damaged due to income dilution or imbalance in asset ratios.

```
// src/bonding_curve/src/swap/pool.rs
pub fn sats_swap_runes(&mut, arg: SatsSwapRunesArg) -> Result<( u128, u128,
u128), String> {
    ......
    let new_sats = self
        .sats
        .checked_add(arg.sats)
        .ok_or("overflow".to_string())?;
    ......
}
```

**Impact:**

An imbalance in the asset ratio of the liquidity pool will exacerbate impermanent loss (LPs suffer additional losses due to the deviation of pool prices from market fair value); LP returns are diluted or arbitrageurs manipulate pool assets through false transactions, resulting in the failure of the incentive mechanism; protocol fees as core income are locked in the pool, which may cause token inflation (such as over-issuance of rewards) or a break in the capital chain (the protocol loses operating funds); complex calculations may also introduce smart contract vulnerabilities (such as incorrect product constants), exacerbating systemic risks.

**Recommended Mitigation:**

Separate transaction fees and LP returns from the Swap Pool. Transaction fees are directly deposited into the protocol treasury for ecological incentives or security funds, and LP returns are distributed through independent accounts (such as staking contracts). A dynamic adjustment mechanism is introduced (such as automatically adjusting the injection amount based on the proportion of assets in the pool). The flow of funds is transparently disclosed on the chain (such as publicizing each injection record through event logs), and abnormal injection operations are restricted through multi-signature governance.

NovalCLabs: Fixed at 1b09418

VelintSec: Confirmed

### 7.1.4 VLS_FMW_004: Unsafe Media Upload / Metadata Injection Vulnerability

**Description:**

The is_logo_valid function only checks the decoded base64 logo data size but fails to validate its content type or sanitize malicious payloads. Attackers can craft a base64 string containing JavaScript code (e.g., disguised as data:text/html;base64,...) or executable scripts. Since the validation skips MIME-type verification (e.g., image/png, image/jpeg), the malicious payload persists in storage and executes when rendered.

```
// src/bonding_curve/src/types.rs
fn is_logo_valid(base64_str: &str, max_size_bytes: usize) -> Result<(), String>
{
    let data_part = base64_str.split(',').nth(1).unwrap_or(base64_str);
    let decoded = STANDARD.decode(data_part)
        .map_err(|e| format!("decoded err: {}", e))?;

    if decoded.len() > max_size_bytes {
        return Err(format!("logo exceeds size limit {:?}", max_size_bytes))
    }
    Ok(())
}
```

**Impact:**

When the logo is rendered as an <img> tag or embedded via src attributes, browsers may execute embedded JavaScript, enabling stored XSS attacks. This allows session hijacking, cookie theft, or redirection to phishing sites. Attackers could also leverage this to trigger server-side malicious file processing or SSRF if the logo data is parsed unsafely.

**Recommended Mitigation:**

- Validate MIME types explicitly by checking the base64 header (e.g., enforce image/* prefixes) and use libraries like infer or magic-number to detect file signatures.
- Sanitize base64 inputs by stripping non-image data and escaping HTML-sensitive characters.
- Frontends should render logos using SafeSrc patterns (e.g., Angular's DomSanitizer) and avoid dangerouslySetInnerHTML (React).
- Implement Content Security Policy (CSP) headers to block inline script execution.

NovalCLabs: Fixed at 57bcbc6 and 132be40

VelintSec: Confirmed

### 7.1.5 VLS_FMW_005: Insecure URL Validation Vulnerability

**Description:**

The is_url_valid() function performs only basic checks on URL length and format but fails to validate critical security aspects. It does not restrict allowed domains, block dangerous URL schemes (e.g., javascript:, data:), or prevent malicious redirects (e.g., https://trusted.com?redirect=evil.com). Attackers can exploit this by submitting URLs leading to phishing sites, SSRF endpoints, or malicious scripts.

```
// src/bonding_curve/src/types.rs
fn is_url_valid(url_str: &str) -> Result<(),String> {
    if url_str.len() > 2000 {
        return Err("url too long".into())
    }
    if let Err(_) = Url::parse(url_str) {
        return Err("Illegal url".into())
    }
    Ok(())
}
```

**Impact:**

This flaw enables Open Redirect attacks (tricking users into visiting malicious sites), Server-Side Request Forgery (SSRF) (allowing attackers to probe/internal network access), and phishing (via fake login pages). If the URL is rendered in frontends (e.g., <img src>), stored XSS may also occur if unsafe schemes (like data:) are permitted

**Recommended Mitigation:**

- Strict URL Validation: Use an allowlist of trusted domains (e.g., allowed_domains = ["cdn.example.com", "trusted-storage.org"]). Block dangerous schemes (javascript:, data:, file:, internal IPs).
- Sanitize Redirects: Reject URLs containing redirect_to, next, or similar parameters.
- When a user clicks a link to navigate to a cross-origin site, the security mechanism should use a pop-up to inform the user to check the link

NovalCLabs: Fixed at de2f595 and c99d4f6

VelintSec: Confirmed

### 7.1.6 VLS_FMW_006: Unsafe URL Validation Leading to Open Redirect & SSRF

**Description:**

In the process of creating meme tokens, there was a lack of security checks on the website field. The is_url_valid() function performs basic length and format checks but fails to validate critical security aspects of the URL. It doesn't restrict allowed domains, block dangerous schemes (e.g., javascript:, data:), or prevent malicious redirect parameters (e.g., ?redirect=evil.com). Attackers can exploit this to submit URLs leading to phishing sites, internal SSRF endpoints, or malicious scripts..

```rust
// src/bonding_curve/src/types.rs
impl CreateMemeTokenArg {
    pub fn check(&self) -> Result<(), String> {
        //......
        if let Some(website) = self.website.clone() {
            is_url_valid(&website)?;
        }
        Ok(())
    }
}
fn is_url_valid(url_str: &str) -> Result<(),String> {
    if url_str.len() > 2000 {
        return Err("url too long".into())
    }
    if let Err(_) = Url::parse(url_str) {
        return Err("Illegal url".into())
    }
    Ok(())
}
```

**Impact:**

This vulnerability enables Open Redirect attacks (tricking users into visiting malicious sites), Server-Side Request Forgery (SSRF) (accessing internal systems), and phishing (via fake login pages). If the URL is rendered in frontends (e.g., as a clickable link), it could also facilitate CSRF or stored XSS if unsafe schemes are allowed.

**Recommended Mitigation:**

- Strict URL Validation: Use an allowlist of trusted domains (e.g., ["trusted.com"]). Block dangerous schemes (javascript:, data:, file:).
- Parameter Sanitization: Reject URLs containing redirect, next, or similar parameters.
- When a user clicks a link to navigate to a cross-origin site, the security mechanism should use a pop-up to inform the user to check the link

NovaICLabs: Fixed at 5d4963a and c99d4f6

VelintSec: Confirmed

## 7.2 Medium Risk

### 7.2.1 VLS_FMW_007: Stored XSS via Unsanitized Token Description

**Description:**

During the creation of meme tokens, only the length check was performed on the description field, but fails to sanitize or escape HTML/JavaScript content. Attackers can inject malicious scripts (e.g., <script>alert(1)</script>) or CSRF payloads (e.g., <img src="http://attacker.com/steal-cookie">) into the description. When rendered in a frontend application without proper escaping, this leads to client-side code execution.

```
// src/bonding_curve/src/types.rs
impl CreateMemeTokenArg {
    pub fn check(&self) -> Result<(), String> {
        //......
        if self.description.chars().count() > 100 {
            return Err("description no longer than 100 characters".into());
        //......
        Ok(())
    }
}
```

**Impact:**

This vulnerability enables stored Cross-Site Scripting (XSS), allowing attackers to:

- Steal user sessions/cookies via JavaScript.
- Perform unauthorized actions (CSRF) if combined with state-changing requests.
- Deface the platform or redirect users to phishing sites.
- If the description is displayed in admin panels, privilege escalation may also occur.

**Recommended Mitigation:**

- Input Sanitization: Use libraries like OWASP Java Encoder (Java) or DOMPurify (JavaScript) to strip HTML/JS. Enforce plaintext-only descriptions or whitelist safe HTML tags (e.g., <b>, <i>).
- Output Encoding: Frontends should render descriptions with safe methods (e.g., React's dangerouslySetInnerHTML avoidance, Angular's {{}} auto-escaping).
- Validation Enhancement: Reject descriptions containing <, >, javascript:, or onerror= patterns.

NovaICLabs: Fixed at d16e2f3

VelintSec: Confirmed

### 7.2.2 VLS_FMW_008: Stored XSS via Unsanitized Twitter Handle Field

**Description:**

The twitter field validation in CreateMemeTokenArg only checks the input length but fails to sanitize or escape special characters. Attackers can inject malicious scripts (e.g., <script>alert(1)</script>) or CSRF payloads (e.g., <img src="attacker.com/steal-cookie">) into the Twitter handle. When rendered unsafely in frontend applications, this leads to client-side code execution

```
// src/bonding_curve/src/types.rs
impl CreateMemeTokenArg {
    pub fn check(&self) -> Result<(), String> {
        //......
        if let Some(twitter) = self.twitter.clone() {
            if twitter.len() > 2000 {
                return Err("Illegal twitter".into())
            }
        //......
        Ok(())
    }
}
```

**Impact:**

This vulnerability enables stored Cross-Site Scripting (XSS), allowing attackers to:

- Hijack user sessions by stealing cookies
- Perform unauthorized actions via CSRF
- Deface the platform or redirect to phishing sites
- If displayed in admin interfaces, it could enable privilege escalation.

**Recommended Mitigation:**

- Input Sanitization: Use libraries like DOMPurify to strip HTML/JS. Enforce strict regex patterns (e.g., ^@?[A-Za-z0-9_]{1,15}$ for Twitter handles).
- Output Encoding: Frontends should render with safe methods (React's {twitterHandle}, Vue's {{ }}).
- Validation Enhancement: Reject inputs containing <, >, ", ', or “javascript:”.

NovaICLabs: Fixed at 3f1e996

VelintSec: Confirmed

### 7.2.3 VLS_FMW_009: Stored XSS via Unsanitized Telegram Handle Field

**Description:**

The telegram field validation in CreateMemeTokenArg only verifies input length without sanitizing special characters or validating content. Attackers can inject malicious scripts (e.g., <script>alert(1)</script>) or CSRF payloads (e.g., <img src="attacker.com/steal-

cookie">) into the Telegram field. When rendered unsafely in frontend interfaces, this enables client-side code execution.

```
// src/bonding_curve/src/types.rs
impl CreateMemeTokenArg {
    pub fn check(&self) -> Result<(), String> {
        //......
        if let Some(telegram) = self.telegram.clone() {
            if telegram.len() > 2000 {
                return Err("Illegal telegram".into())
            }
        }
        //......
        Ok(())
    }
}
```

**Impact:**

This vulnerability introduces stored Cross-Site Scripting (XSS) risks, allowing attackers to:

- Steal user sessions via cookie theft
- Perform unauthorized actions through CSRF
- Deface platforms or redirect to malicious sites
- If displayed in admin panels, it may facilitate privilege escalation.

**Recommended Mitigation:**

- Input Sanitization: Use libraries like DOMPurify to strip HTML/JS. Enforce strict regex (e.g., ^@?[a-zA-Z0-9_]{5,32}$ for Telegram handles).
- Output Encoding: Frontends should render with safe methods (e.g., React's {telegramHandle}).
- Validation Rules: Reject inputs containing <, >, ", ', or "javascript:".

NovalCLabs: Fixed at b149894

VelintSec: Confirmed

## 7.3 Low Risk

### 7.3.1 VLS_FMW_010: Insufficient error handling

**Description:**

The internal_transfer_to_external function lacks transactional safety and error recovery mechanisms:

- No Rollback on Failure: If icrc1_transfer fails (e.g., network error or insufficient balance), the subsequent burn operation proceeds unconditionally, permanently destroying funds without completing the transfer.
- Non-Atomic Operations: The two-step process (transfer + burn) lacks atomicity, risking partial execution and inconsistent state (e.g., tokens burned but not transferred).
- Silent Failures: Missing contextual logging makes debugging and auditing failures (e.g., intermittent API errors) nearly impossible.

```rust
// src\bonding_curve\src\internal_ledger\ledger.rs
pub async fn internal_transfer_to_external(meme_token: &MemeToken) ->
Result<(), String> {
    let to = meme_token.get_account();
    let icrc1_token = meme_token.clone().bc.token;
    let internal_account = meme_token.get_account();
    let ledger_type = LedgerType::ICRCToken(icrc1_token.canister_id);
    let internal_balance = internal_balance_of(ledger_type.clone(),
internal_account);

    // Situations where icrc1_transfer may fail are not handled
    icrc1_transfer(icrc1_token.canister_id, TransferArg{...}).await?;

    // If BURN fails, funds may be lost
    burn(ledger_type, meme_token.get_account(), internal_balance)?;

    Ok(())
}
```

**Impact:**

These flaws threaten fund integrity and system reliability:

- Irreversible Fund Loss: A failed icrc1_transfer followed by a successful burn permanently removes tokens from the internal account without external delivery.
- State Corruption: Partially executed workflows leave balances mismatched, enabling exploits like double-spending or ghost liabilities.
- Opaque Debugging: Without logs or transaction IDs, operators cannot trace root causes of intermittent failures (e.g., subnet congestion).

**Recommended Mitigation:**

- Implement Transactional Rollbacks: Execute burn only after confirming icrc1_transfer success. Add compensatory actions (e.g., minting back burned tokens) if later steps fail.
- Introduce Retry Logic: Retry transient errors (e.g., network timeouts) with exponential

backoff.

- Track Transactions: Generate unique IDs for each operation to trace cross-canister calls.

**NovaICLabs:** Fixed at 1876bf6

**VelintSec:** Confirmed