LEDGEROPS

SECURITY ASSESSMENT REPORT

Solve3

January 23, 2025

Validation: April 25, 2025

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This report presents the findings from a comprehensive security assessment conducted on Solve3 Finance's platform, encompassing the Solve3 Web Application, smart contracts, and their integration with the ORCA-SO Whirlpools framework. The assessment aimed to evaluate the platform's security posture, identify vulnerabilities, and provide actionable insights to mitigate potential risks. The engagement was performed by LedgerOps between January 6, 2025, and January 10, 2025, using a combination of automated tools and manual techniques.

Scope of Testing

The security assessment covered the following components:

1. **Web Application**: Evaluation of Solve3's user interface and API endpoints.
2. **Smart Contracts**: Assessment of critical functionalities, including vote escrow mechanisms (ve33), token management (Solve3), liquidity/swaps (Solve3fi), and token initialization (xSolve3), along with their integration with the ORCA-SO Whirlpools framework.

Risk Summary

A total of **0 Critical**, **2 High**, **2 Medium**, **1 Low**, and **1 Informational** findings were identified. If exploited, these vulnerabilities could result in financial loss, unauthorized access, or operational disruptions.

Important Note

To maintain a secure environment, regular monitoring, timely updates to software and dependencies, and comprehensive end-user training are essential. Additionally, regular reviews of network segments, hosts, and critical systems should be conducted to ensure a robust security posture.

REPORT CARD

Web Application

Finding	Status	Severity
Information Disclosure	**Closed**	**Low**
Invalid Characters Error in Token Swap	**Closed**	**Informational**

Smart Contracts

Finding	Status	Severity
Missing Access Controls	**Closed**	**High**
Unrestricted Function Calls	**Closed**	**High**
Unchecked Math Operations	**Closed**	**Medium**
Incomplete Balance Verification	**Closed**	**Medium**

ENGAGEMENT SCOPE

SCOPE OF WORK

This report is part of the security assessment conducted during the period outlined in the timeline and covers the security assessment scopes of concern for Solve3.

TESTING SCOPE

Here's pertinent information about the testing scope. We believe you'll find this information about the testing scope to be pertinent, as that is exactly the reason it was judged worthy of inclusion, in this particular portion of the report narrative.

ASSUMPTIONS

NDA and rules of engagement have been signed based on the information provided and gathered during information gathering phase and the company name is Solve3.

TIMELINE

The timeline of the testing is included below:

Test Type	Start Date/Time	End Date/Time
Security Assessment	January 6, 2025	January 10, 2025
Validation Assessment	April 21, 2025	April 25, 2025

TECHNICAL FINDINGS OVERVIEW (WEB APPLICATION)

	Critical	High	Medium	Low	Informational
Web Application	0	0	0	1	1

The Solve3 web application contains vulnerabilities that could impact both the security of the application and the reputation of Solve3 Finance. Addressing these issues is critical to ensuring a secure production environment and maintaining user trust.

LedgerOps strongly recommends remediating these vulnerabilities prior to deployment in a live environment. Failure to do so could expose the platform to unauthorized access, data breaches, and operational disruptions.

For detailed technical information on all identified risks, including their severity, evidence, and recommended remediation steps, please refer to the Technical Findings section of this report. Below is a summary of the vulnerabilities identified during the assessment.

Finding Name	Risk Level	
Low Risk Findings		
Information Disclosure	Low	CVSS 3.7
Informational Risk Findings		
Invalid Characters Error in Token Swap	Informational	N/A

INFORMATION DISCLOSURE

Low

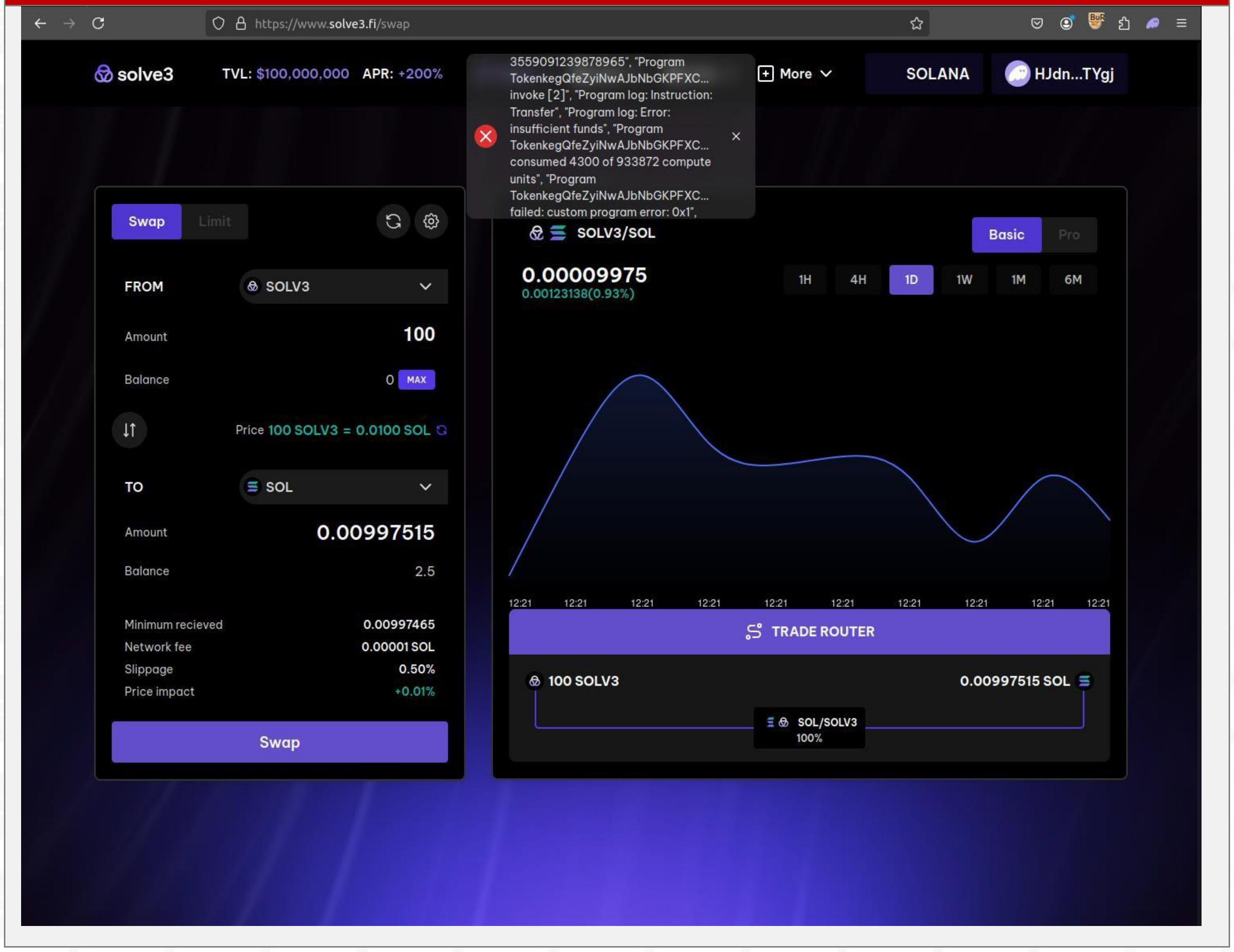
SUMMARY

The application displays verbose error messages to users, revealing unnecessary details about backend logic and system configuration. This information could assist attackers in identifying potential vulnerabilities by exposing details such as program logs, compute units, and error codes.

EVIDENCE

During testing, invalid actions or unexpected inputs triggered verbose error messages. The screenshot below demonstrates a case where internal details such as program logs and consumed compute units were exposed:

Example



```
3559091239878965", "Program
TokenkegQfeZyiNwAJbNbGKPFXC...
invoke [2]", "Program log: Instruction:
Transfer", "Program log: Error:
insufficient funds", "Program
TokenkegQfeZyiNwAJbNbGKPFXC...
consumed 4300 of 933872 compute
units", "Program
TokenkegQfeZyiNwAJbNbGKPFXC...
failed: custom program error: 0x1",
```

REMEDIATION

- Configure the application to display generic error messages to users, such as “An error occurred. Please try again.”
- Log detailed error messages server-side for debugging purposes, ensuring they are not exposed to the end user.
- Regularly review error handling mechanisms to verify that sensitive information is not unintentionally disclosed.

VALIDATION

- **Status:** Remediated / Closed
- **Date:** April 23, 2025

CVSS 3.7
AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:N/A:N

INVALID CHARACTERS ERROR IN TOKEN SWAP

Informational

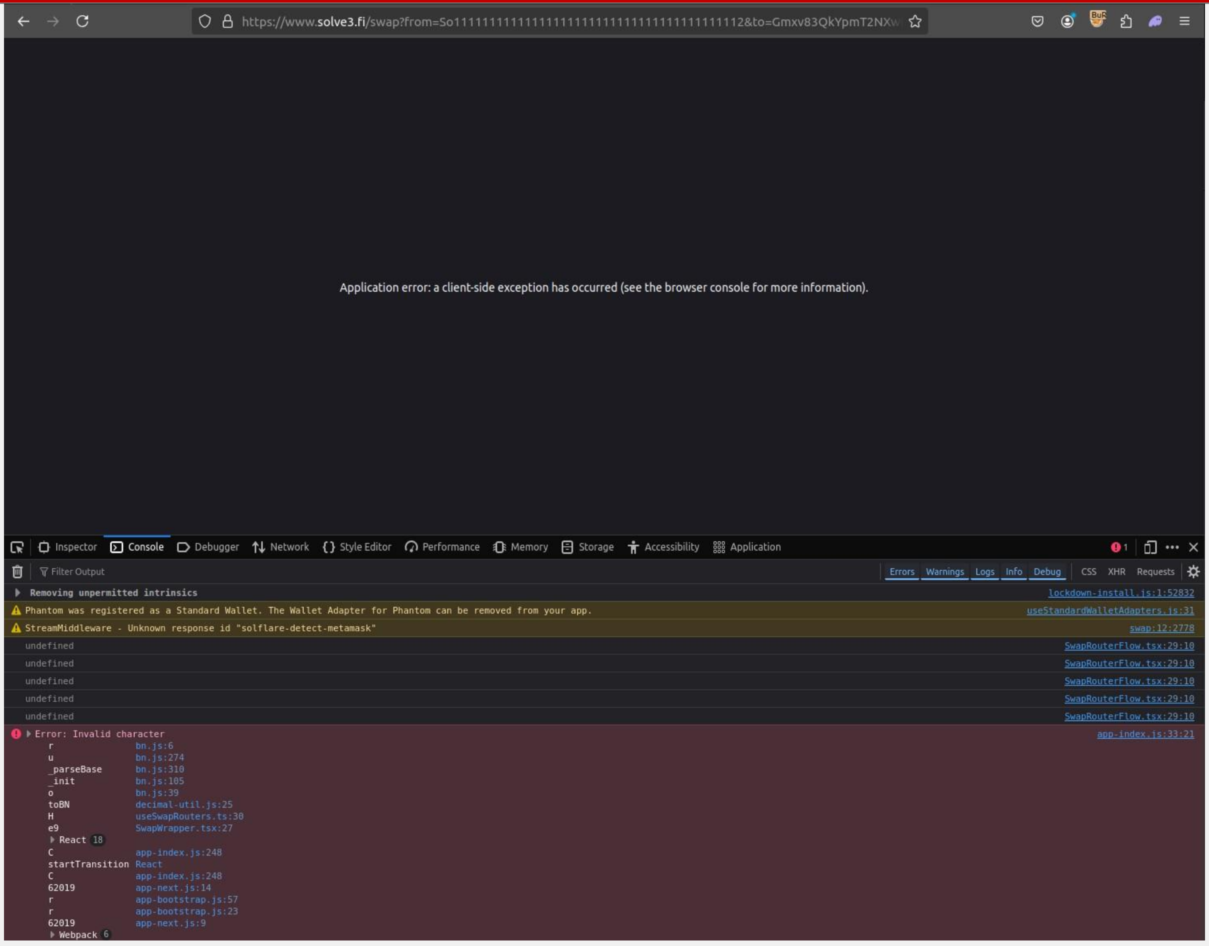
SUMMARY

When swapping tokens, the application errors out if a period (e.g., .) is entered in the amount field. This indicates insufficient input validation and can disrupt user experience. While this issue does not present a direct security risk, it highlights potential gaps in input handling that could lead to broader issues.

EVIDENCE

During testing, entering a period as part of the token swap amount resulted in a client-side application error. The screenshot below demonstrates the error triggered in the application:

Example



Application error: a client-side exception has occurred (see the browser console for more information).

```
Removing unpermitted intrinsics                                                      lockdown-install.js:1:52832
Phantom was registered as a Standard Wallet. The Wallet Adapter for Phantom can be removed from your app.   useStandardWalletAdapters.js:31
StreamMiddleware - Unknown response id "solflare-detect-metamask"                    swap:12:2778
undefined                                                                            SwapRouterFlow.tsx:29:10
undefined                                                                            SwapRouterFlow.tsx:29:10
undefined                                                                            SwapRouterFlow.tsx:29:10
undefined                                                                            SwapRouterFlow.tsx:29:10
undefined                                                                            SwapRouterFlow.tsx:29:10
Error: Invalid character                                                             app-index.js:33:21
    r               bn.js:6
    u               bn.js:274
    _parseBase      bn.js:310
    _init           bn.js:105
    o               bn.js:39
    toBN            decimal-util.js:25
    H               useSwapRouters.ts:30
    e9              SwapWrapper.tsx:27
    React 18
    C               app-index.js:248
    startTransition React
    C               app-index.js:248
    62019           app-next.js:14
    r               app-bootstrap.js:57
    r               app-bootstrap.js:23
    62019           app-next.js:9
    Webpack 6
```

REMEDIATION

- Implement strict input validation for the token swap field to ensure only valid numerical formats (e.g., integers or decimals) are accepted.
- Display user-friendly error messages to inform users of invalid input, rather than exposing technical errors.
- Conduct comprehensive input sanitization and validation at both client and server levels to ensure consistency.

VALIDATION

- **Status:** Remediated / Closed
- **Date:** April 23, 2025

CVSS N/A

TECHNICAL FINDINGS OVERVIEW (SMART CONTRACTS)

	Critical	High	Medium	Low	Informational
Smart Contracts	0	2	2	0	0

The smart contracts within Solve3 Finance's platform, including their integration with the ORCA-SO Whirlpools framework, contain vulnerabilities that could impact the integrity of the system, lead to financial loss, and damage the reputation of the organization. While the ORCA-SO Whirlpools framework provides certain security assurances, these dependencies require regular review to ensure they do not introduce additional risks to the platform.

LedgerOps strongly recommends remediating the identified vulnerabilities prior to deployment in a live environment. Failure to address these issues could result in exploitation by malicious actors, leading to unauthorized access, financial compromise, or disruption of core functionalities.

For detailed technical information on all identified risks, including their severity, evidence, and recommended remediation steps, please refer to the Technical Findings section of this report. Below is a summary of the vulnerabilities identified during the assessment.

Finding Name	Risk Level	
High Risk Findings		
Missing Access Control	High	
Unrestricted Function Call	High	
Medium Risk Findings		
Incomplete Balance Verification	Medium	
Unchecked Math Operation	Medium	

MISSING ACCESS CONTROL

High

SUMMARY

LedgerOps determined that the *initialize_lock* function does not enforce explicit access control, allowing unauthorized actors to execute this function. Without role-based restrictions, an attacker could call this function and set themselves as the *lock_authority*, effectively gaining control over token locks.

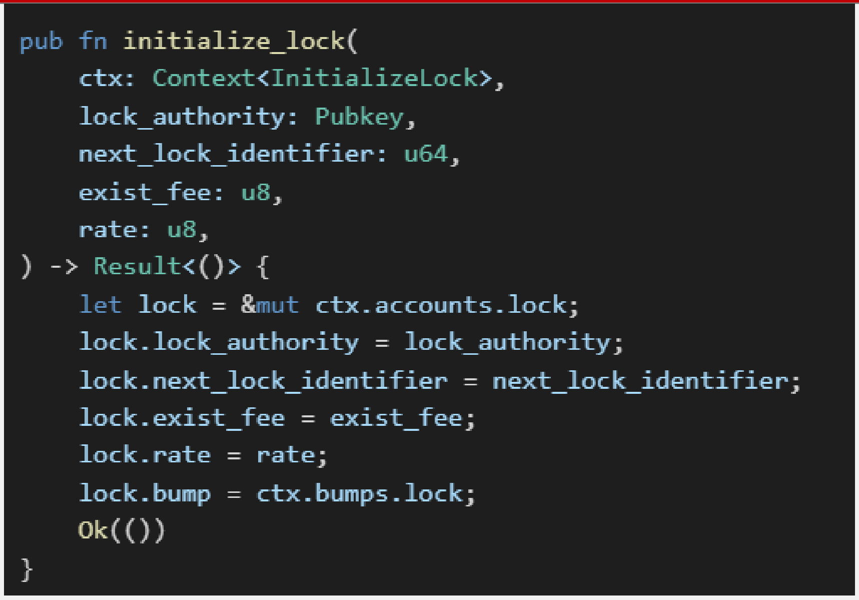
If *initialize_lock* is intended to be called only by privileged users (e.g., admins or governance-controlled entities), this could lead to unauthorized token locking, disrupting the protocol and potentially locking up user assets indefinitely.

AFFECTED FILES

- /programs/ve33/src/instructions/initialize_lock.rs
- /programs/ve33/src/lib.rs

EVIDENCE

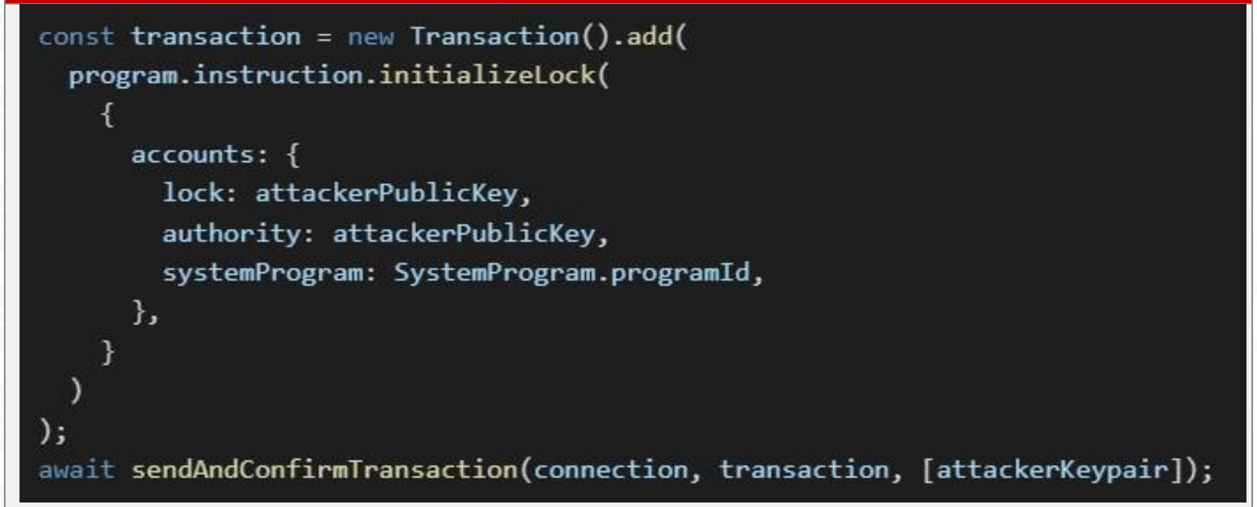
Vulnerable code in initialize_lock.rs



```rust
pub fn initialize_lock(
    ctx: Context<InitializeLock>,
    lock_authority: Pubkey,
    next_lock_identifier: u64,
    exist_fee: u8,
    rate: u8,
) -> Result<()> {
    let lock = &mut ctx.accounts.lock;
    lock.lock_authority = lock_authority;
    lock.next_lock_identifier = next_lock_identifier;
    lock.exist_fee = exist_fee;
    lock.rate = rate;
    lock.bump = ctx.bumps.lock;
    Ok(())
}
```

An attacker can exploit this issue by invoking initialize_lock without restriction, allowing them to set themselves as the lock authority. This could enable unauthorized users to control locked assets, preventing legitimate users from accessing their funds. The following Solana Web3 SDK code demonstrates how an attacker can execute the function programmatically:

Proof of Concept Exploit



```
const transaction = new Transaction().add(
  program.instruction.initializeLock(
    {
      accounts: {
        lock: attackerPublicKey,
        authority: attackerPublicKey,
        systemProgram: SystemProgram.programId,
      },
    }
  )
);
await sendAndConfirmTransaction(connection, transaction, [attackerKeypair]);
```

REMEDIATION

LedgerOps recommends applying an access control macro to enforce role-based restrictions on this function. This will ensure that only privileged users can initialize locks, preventing unauthorized token locking. Alternatively, implementing a manual check to verify only authorized roles can execute it would mitigate this risk.

Sample remediation code

```
#[access_control(admin_only(ctx.accounts.funder))]
```

VALIDATION

- **Status:** Remediated / Closed
- **Date:** April 23, 2025

UNRESTRICTED FUNCTION CALL

High

SUMMARY

LedgerOps determined that the *swap* function lacks access control, allowing any user to execute swaps without restrictions. This opens up the potential for price manipulation attacks where users could repeatedly call the function to exploit rounding errors or fee miscalculations.

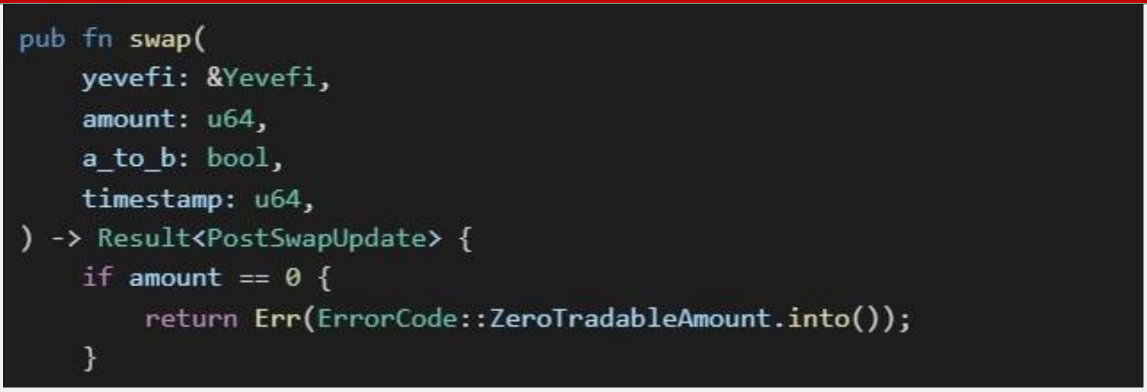
A malicious user could initiate swaps with manipulated amounts, draining liquidity pools or front-running transactions. This could result in market manipulation, where attackers profit from artificially inflated or deflated token prices at the expense of legitimate traders.

AFFECTED FILES

- /programs/solve3/src/manager/swap_manager.rs

EVIDENCE

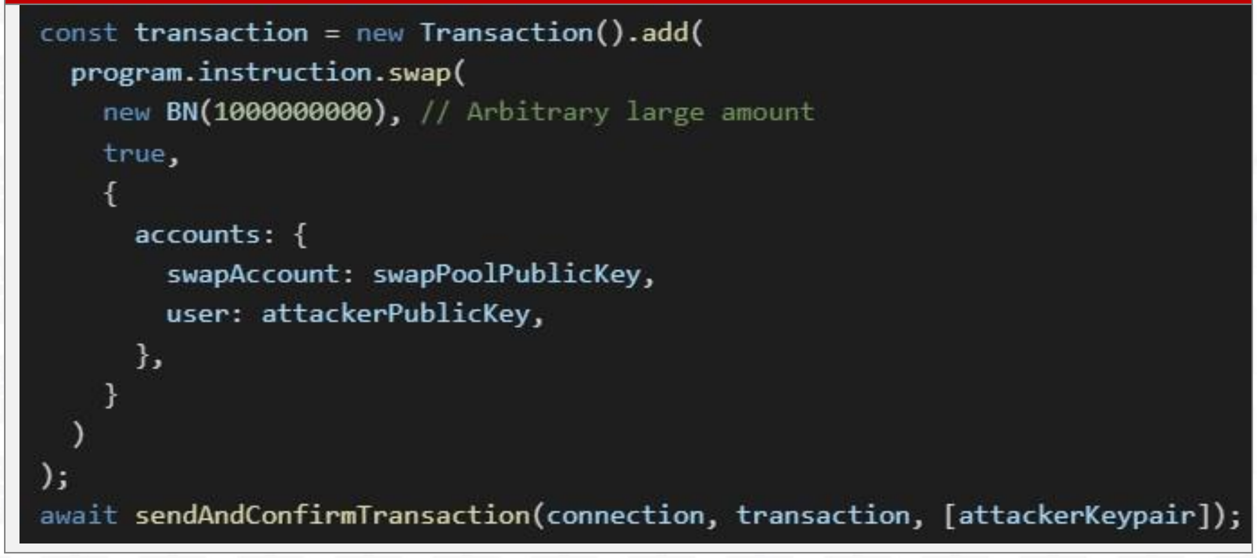
Vulnerable code in swap_manager.rs



```
pub fn swap(
    yevefi: &Yevefi,
    amount: u64,
    a_to_b: bool,
    timestamp: u64,
) -> Result<PostSwapUpdate> {
    if amount == 0 {
        return Err(ErrorCode::ZeroTradableAmount.into());
    }
```

An attacker can repeatedly call swap with manipulated values to drain liquidity pools or front-run transactions. By executing multiple swaps with large amounts, they can manipulate token prices and execute arbitrage at the expense of legitimate traders. The following code shows how an attacker could execute a large swap transaction:

Proof of Concept Exploit



```
const transaction = new Transaction().add(
  program.instruction.swap(
    new BN(1000000000), // Arbitrary large amount
    true,
    {
      accounts: {
        swapAccount: swapPoolPublicKey,
        user: attackerPublicKey,
      },
    }
  )
);
await sendAndConfirmTransaction(connection, transaction, [attackerKeypair]);
```

REMEDIATION

LedgerOps recommends implementing access control to restrict execution to authorized users. This will prevent unauthorized market manipulation and abuse of the swap function. Additionally, introducing rate-limiting mechanisms can mitigate abusive swap behaviors, such as frequent repeated calls to the function within a short period.

Sample remediation code

```
#[access_control(admin_only(ctx.accounts.authority))]
```

VALIDATION

- **Status:** Remediated / Closed
- **Date:** April 23, 2025

INCOMPLETE BALANCE VERIFICATION

Medium

SUMMARY

LedgerOps determined that the *transfer_token* function does not explicitly check if the source account has a sufficient balance before initiating a transfer. This oversight could result in transactions failing midway, leading to inefficiencies and unnecessary fees for users attempting to transfer more tokens than available.

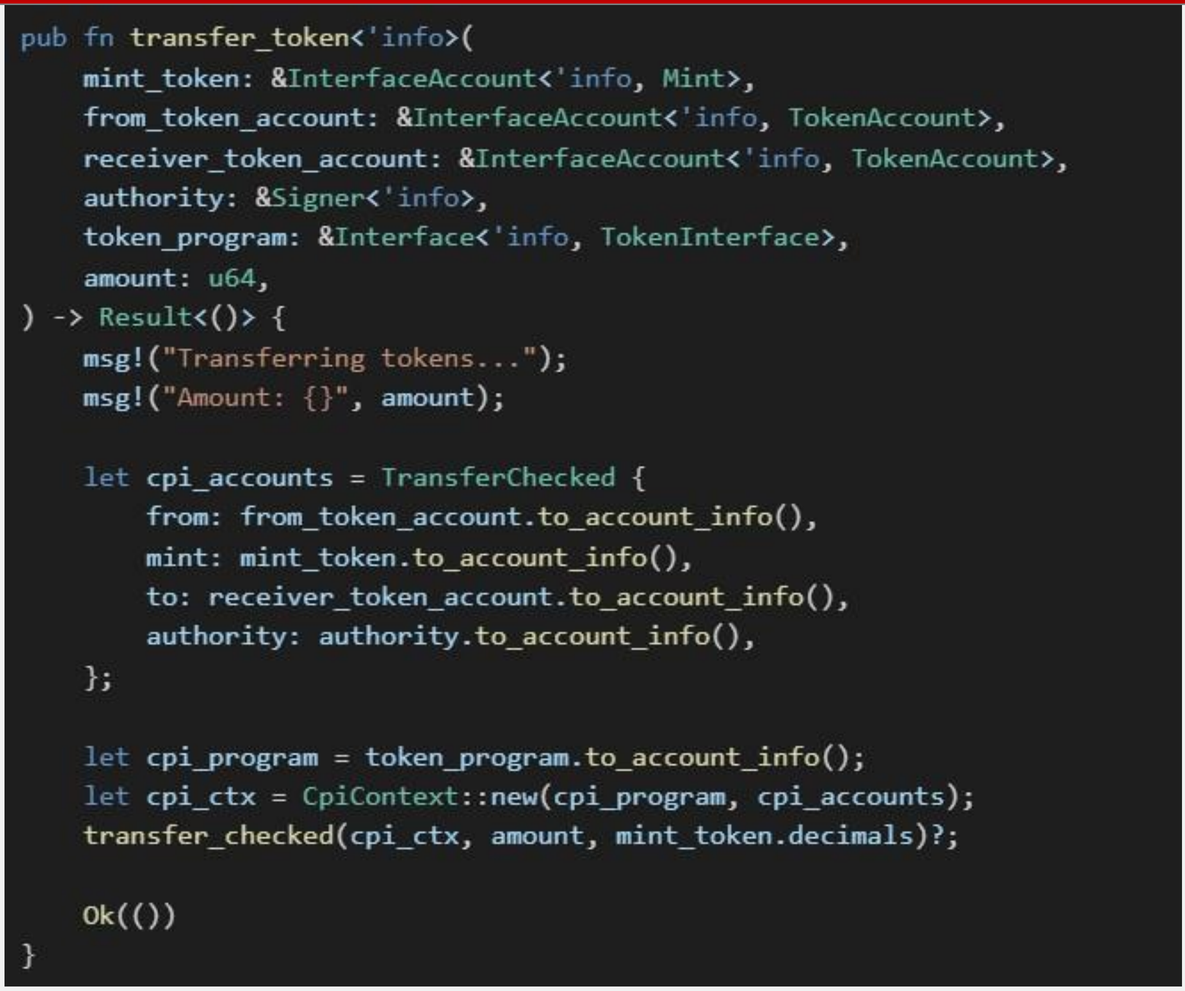
If *transfer_checked* is called with an insufficient balance, the transaction will fail at execution time rather than being preemptively blocked. This could allow for transaction spam attacks, leading to network congestion and potential denial-of-service (DoS) scenarios.

AFFECTED FILES

- /programs/ve33/src/utils/token.rs

EVIDENCE

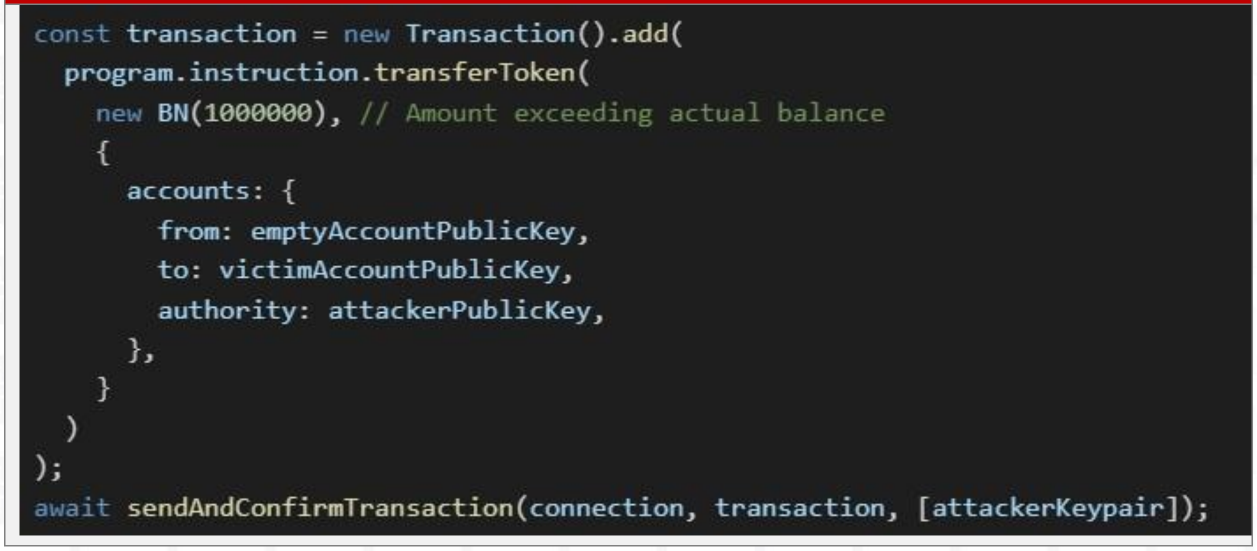
Vulnerable code in token.rs



```rust
pub fn transfer_token<'info>(
    mint_token: &InterfaceAccount<'info, Mint>,
    from_token_account: &InterfaceAccount<'info, TokenAccount>,
    receiver_token_account: &InterfaceAccount<'info, TokenAccount>,
    authority: &Signer<'info>,
    token_program: &Interface<'info, TokenInterface>,
    amount: u64,
) -> Result<()> {
    msg!("Transferring tokens...");
    msg!("Amount: {}", amount);

    let cpi_accounts = TransferChecked {
        from: from_token_account.to_account_info(),
        mint: mint_token.to_account_info(),
        to: receiver_token_account.to_account_info(),
        authority: authority.to_account_info(),
    };

    let cpi_program = token_program.to_account_info();
    let cpi_ctx = CpiContext::new(cpi_program, cpi_accounts);
    transfer_checked(cpi_ctx, amount, mint_token.decimals)?;

    Ok(())
}
```

An attacker can exploit this issue by crafting a transaction that attempts to transfer more tokens than are available in their account. Since the function does not validate the sender's balance beforehand, it will fail only at execution time, potentially causing network congestion or spam attacks. The following code shows how an attacker could attempt to execute such a transaction:

Proof of Concept Exploit



```
const transaction = new Transaction().add(
  program.instruction.transferToken(
    new BN(1000000), // Amount exceeding actual balance
    {
      accounts: {
        from: emptyAccountPublicKey,
        to: victimAccountPublicKey,
        authority: attackerPublicKey,
      },
    }
  )
);
await sendAndConfirmTransaction(connection, transaction, [attackerKeypair]);
```

REMEDIATION

LedgerOps recommends adding an explicit balance check before the transfer to prevent execution failures due to insufficient funds. This will enhance user experience by preventing unnecessary transaction fees and network congestion. Implementing logging or error handling will also notify users before submitting transactions with inadequate balances.

Sample remediation code

```
if from_token_account.amount < amount {
    return Err(ProgramError::InsufficientFunds.into());
}
```

VALIDATION

- **Status:** Remediated / Closed
- **Date:** April 23, 2025

UNCHECKED MATH OPERATION

Medium

SUMMARY

LedgerOps determined that some calculations use unchecked arithmetic (*unchecked_add*, *unchecked_sub*), potentially leading to integer overflows. If these unchecked operations result in an overflow, they could cause liquidity miscalculations, allowing attackers to drain liquidity pools or manipulate pool values.

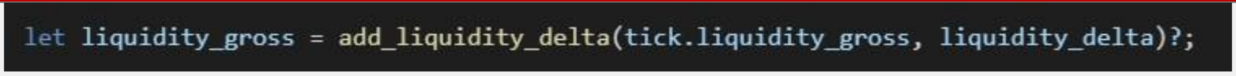
A liquidity calculation error could allow users to withdraw more funds than they originally deposited, ultimately leading to protocol insolvency. Furthermore, an attacker could exploit this weakness to manipulate liquidity calculations and disrupt trading mechanisms.

AFFECTED FILES

- /programs/solve3/src/manager/tick_manager.rs

EVIDENCE

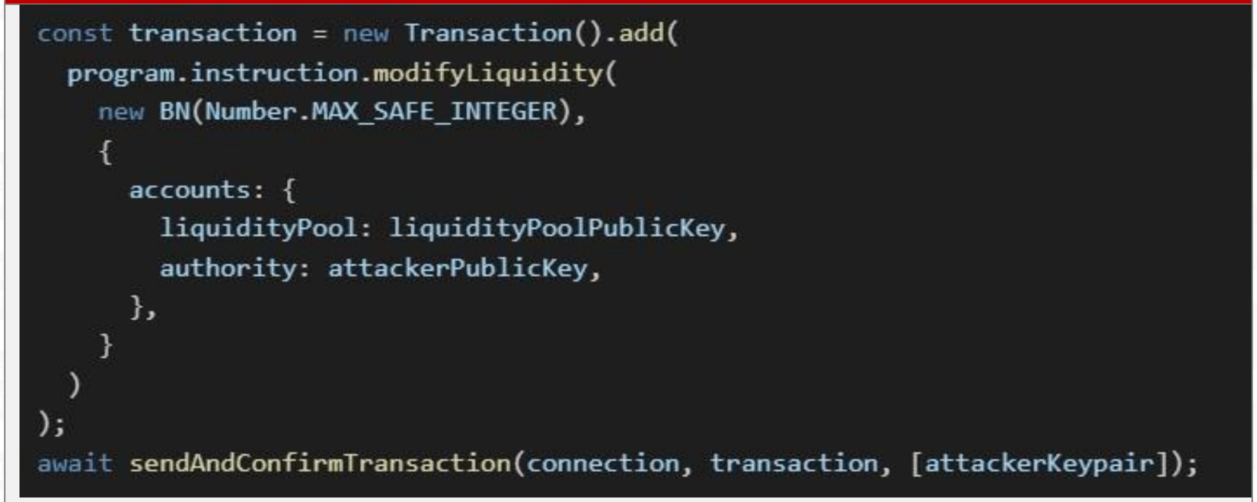
Vulnerable code in tick_manager.rs



```
let liquidity_gross = add_liquidity_delta(tick.liquidity_gross, liquidity_delta)?;
```

An attacker can exploit this issue by providing an extremely large liquidity delta when modifying liquidity. Since the calculation does not use checked arithmetic, this could overflow internal calculations, leading to incorrect liquidity amounts or even allowing users to withdraw excess funds. The following transaction demonstrates how an attacker can attempt to exploit this vulnerability:

Proof of Concept Exploit



```
const transaction = new Transaction().add(
  program.instruction.modifyLiquidity(
    new BN(Number.MAX_SAFE_INTEGER),
    {
      accounts: {
        liquidityPool: liquidityPoolPublicKey,
        authority: attackerPublicKey,
      },
    }
  )
);
await sendAndConfirmTransaction(connection, transaction, [attackerKeypair]);
```

REMEDIATION

LedgerOps recommends using Rust's checked arithmetic functions to prevent overflows. This ensures that calculations do not result in unintended liquidity miscalculations or protocol instability. Additionally, implementing validation checks before applying liquidity updates will help ensure input values remain within safe bounds.

Sample remediation code

```
let liquidity_gross = tick.liquidity_gross
    .checked_add(liquidity_delta)
    .ok_or(ErrorCode::LiquidityOverflow)?;
```

VALIDATION

- **Status:** Remediated / Closed
- **Date:** April 23, 2025