# STX CITY – Bonding Curve

## Security Audit

Final Release Version

Gecko Security – July 27th, 2024

# Table of Contents

# About Gecko Security

Gecko Security is a vulnerability research firm specializing in blockchain security, including EVM, Solana, and Stacks. We assess L1s and L2s, cross-chain protocols, wallets, applied cryptography, zero-knowledge circuits, and web applications. Our team consists of security researchers and software engineers with backgrounds ranging from intelligence agencies to companies like Binance.

Gecko aims to treat clients on a case-by-case basis and to consider their individual, unique business needs. Our services include comprehensive smart contract audits, smart contract fuzzing and formal verification. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. We believe in transparency and actively contribute to the community creating open-source security tools and educational content.

# Overview

## Executive Summary

Gecko Security conducted a security assessment for STX CITY from July 22nd to July 24th, 2024. During this engagement, Gecko reviewed STX CITY's code for security vulnerabilities, design issues and general weaknesses in security posture.

During this audit we found 1 high vulnerability and 2 medium vulnerabilities.

## Scope

Bonding Curve Contracts

| | |
|---|---|
| **Repository** | https://github.com/khoa-nvk/bonding-curve-v1 |
| **Version** | v1: 2cd69873e9411b0daacd6daa04bd3da4c4283720 |
| **Type** | Clarity |
| **Platform** | Stacks |

The audit was based on commit 2cd69873e9411b0daacd6daa04bd3da4c4283720. Fixes were checked on commit 6a3003f7defddab55475d6b69808cf4ae4884921.

## Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon with the client based on priority and outcomes of the audit. In this assessment we sought to answer the following questions:

- Can an attacker gain tokens or STX from the DEX's balance?
- Can tokens be locked due to a denial of service of the DEX?

## Methodology

During a security assessment, Gecko works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Gecko Security was provided with the source code, full stack testnet deployment and documentation that outlines the expected behavior. Our auditors spent 3 days auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. The audit also included adding v1 invariant testing for edge cases and creating a reusable test suite for the contract, to be updated to v2 upon release.

## Findings and Fixes

| ID | Title | Severity | Status |
|---|---|---|---|
| **HI-01** | Token loss due to Malicious Contract | High | Mitigated |
| **ME-01** | Call inside as-contract | Medium | Mitigated |
| **ME-02** | tx-sender in assert | Medium | Resolved |

## Severity Classification

Security risks are classified as follows:

- **Critical**: These are issues we managed to exploit. They compromise the system seriously. They must be fixed immediately.
- **High:** These are significant security vulnerabilities that have a high likelihood of being exploited. They pose a large threat to the system's security and should be addressed promptly.
- **Medium**: These are potentially exploitable issues. Even though we did not manage to exploit them, or their impact is not clear, they might represent a security risk soon. We suggest fixing them as soon as possible.
- **Low**: These issues represent problems that are relatively small or difficult to take advantage of but can be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be considered and be fixed when possible.
- **Enhancement**: These kinds of findings do not represent a security risk. They are best practices that we suggest implementing.

The classification can be summarized as:

| Severity | Exploitable | Roadblock | To be Fixed |
|---|---|---|---|
| **Critical** | Yes | Yes | Immediately |
| **High** | Yes | Yes | As soon as possible |
| **Medium** | In the near future | Yes | Soon |
| **Low** | Unlikely | No | Eventually |
| **Enhancement** | No | No | Optional |

# Findings

## HI-01: Unauthorized Token Loss due to Malicious Contract

A malicious contract can transfer tokens from the DEX's principal to the token owner without the DEX's consent or awareness, leading to the complete loss of all tokens and STX from the DEX's account.

### PoC (Proof of Concept)

The exploit utilizes the '`tx-sender`' keyword to authorize token transactions without the user's explicit consent. For more details, see ME-03.

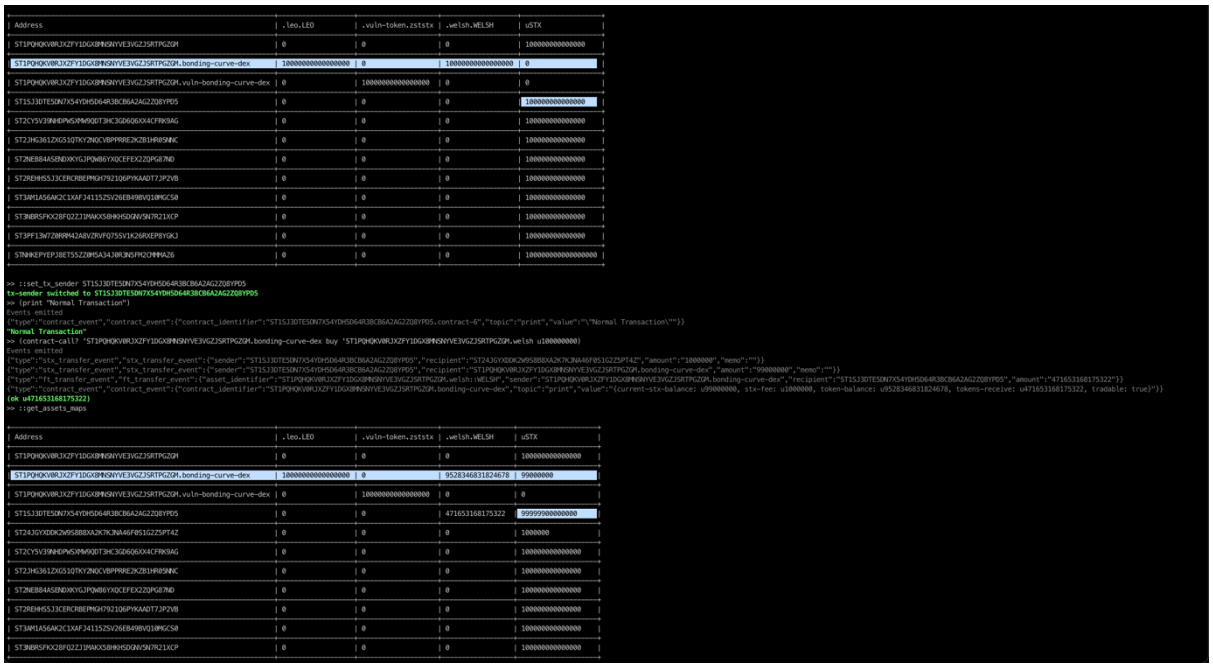


Figure 1: Normal *Buy* transaction of the WELSH token

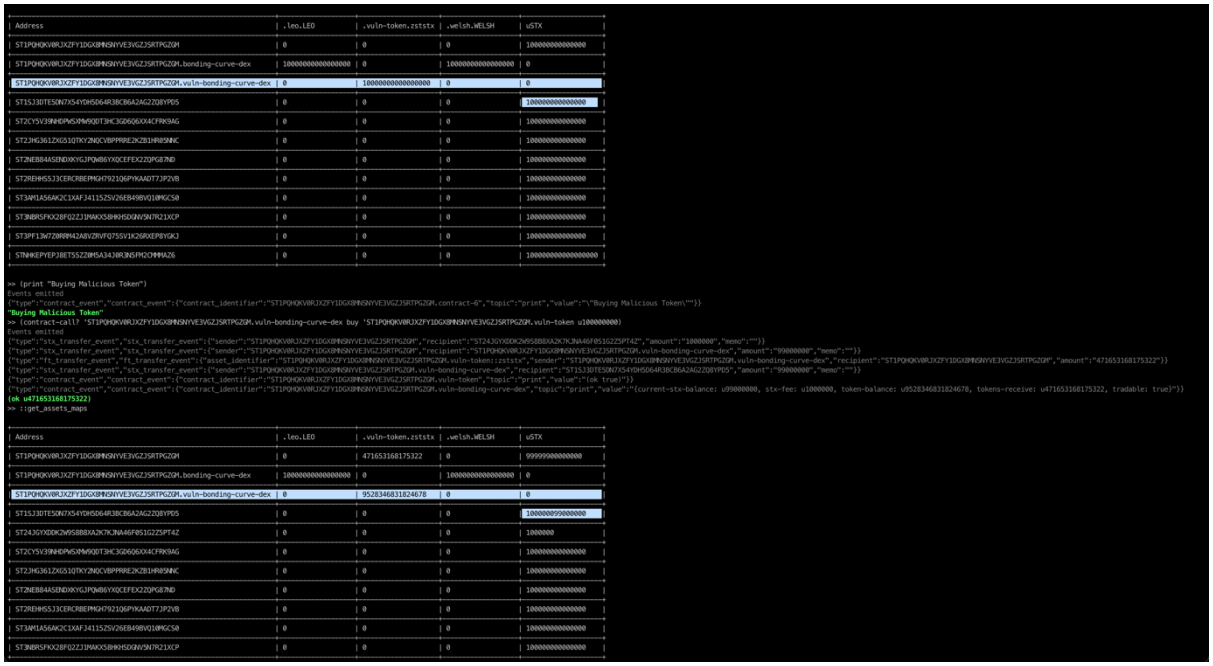


Figure 2: Malicious *Buy* transaction of a vulnerable token (https://github.com/Gecko-Security/bonding-curve-v1/blob/main/contracts/vuln-token.clar)

The malicious contract mimics a normal contract but includes a vulnerable function that transfers the entire token balance of STX and tokens from the caller (the DEX's principal) to the '`TOKEN_OWNER`' upon interaction. This exploit shows that upon buying the token, all the DEX's tokens and STX are sent to the token owner's principal. Steps to reproduce can be found here.

## Remediation

Although STX CITY only allows tokens to be added to the DEX through the frontend application, preventing direct modification to the default contract code, there is a risk an attacker could still connect a malicious token to drain the DEX's funds.

To mitigate this risk, it is recommended to:

- Set post conditions for contract calls, aborting transactions if post conditions fail.
- Use '`contract-caller?`' instead of '`tx-sender`' for tighter security, as it specifically identifies the account or contract making the call.

## Status

**Mitigated:** The frontend reduces exploitability by restricting user modifications to filtered attributes and metadata of a default contract. An attacker would need to further exploit another vulnerability to bypass the web frontend and upload a custom token to the bonding curve. Users can also verify the DEX's contract and token contract on the frontend easily too.

## ME-01: Call inside As-Contract

In Clarity, when executing a contract call, the '`tx-sender`' is the contract that initiated the call. This means that if a contract calls another contract, the '`tx-sender`' of the called contract will be the calling contract. This can be exploited by a malicious contract to impersonate another contract.

### Location

- *contracts/bonding-curve-dex.clar*: 65, 78, 80

### Remediation

Use a pre-whitelisted contract instead of allowing users to pass an arbitrary contract.

### Status

**Mitigated:** To keep a decentralized approach and fair launch of tokens, users are prompted to verify both the token and DEX contracts and protected through frontend safeguards and post conditions.

## ME-02: Authentication via Tx-Sender in Assert

The system uses '`tx-sender`' for authentication, which can expose users to phishing threats. Users might unknowingly activate a malicious contract, allowing it to access and initiate functions, impersonating the original user. This poses risks depending on the accessed function.

Although the system only uses '`tx-sender`' for owner authentication, reducing exploitation risk with a dedicated account, its changing nature during contract execution can lead to unexpected behavior in an '`assert`' statement. This vulnerability can be chained with other exploits, as seen in HI-01.

### Location

- *contracts/bonding-curve-dex.clar*: 116

### Remediation

It is advisable to switch from using '`tx-sender`' to contract-caller for a more reliable and secure authentication method. Furthermore, introducing a mapping for trusted callers can add an extra layer of security, particularly if the system needs to interact with specific intermediary contracts.

### Status

**Resolved:** Fixed according to the given recommendation.

## Invariant Testing

Fuzzing was conducted using property-based testing, defining invariants to specify the desired end state and using random values to test edge cases and complex states. The test suite currently includes v1 invariant testing and will be updated to v2 when ready.

| Invariant | Description | Passed |
|---|---|---|
| BalanceChecks | Checks users can't buy or sell tokens without sufficient balance. | Yes |
| BuyTokens | Checks that buying tokens works correctly including fee calculation and balance updates. | Yes |
| ConstantProduct | Ensures that constant product formula (k=x*y) is true. | Yes |
| Fee Collection | Checks that correct fees are collected. | Yes |
| LiquidityProvision | Checks that liquidity is provided to the DEX when the STX target amount is reached. | Yes |
| PriceDecrease | Checks that the token price decreases when tokens are sold. | Yes |
| PriceIncrease | Checks that the token price increases when tokens are brought. | Yes |
| SellTokens | Checks that selling tokens works correctly, including STX payout and balance updates. | Yes |
| TokenBurning | Checks that tokens are burned correctly when the STX target is reached. | Yes |
| TradingEnabled | Checks that the trading status is correct. | Yes |

# Conclusion

We found the contracts to be simple and straightforward, with sufficient documentation. Gecko Security also added an upgradable testing suite. In the initial testing, we identified 1 high issue and 2 medium issues. The development team has acknowledged these problems and is actively working on fixes, with remediation discussions ongoing.

---

**Disclaimer**: This assessment does not guarantee the discovery of all possible issues within its scope, nor does it ensure the absence of future issues. Additionally, Gecko Security Inc. cannot guarantee the security of any code added to the project after our assessment. Since a single assessment is never comprehensive, we recommend conducting multiple independent assessments and implementing a bug bounty program.

For each finding, Gecko Security Inc. provides a recommended solution. These recommendations illustrate potential fixes but may not be tested or functional. These recommendations are not exhaustive and should be considered a starting point for further discussion. We are available to provide additional guidance and advice as needed.