CoinFabrik

Velar Audit

February 2024

By CoinFabrik

v202311

Executive Summary

CoinFabrik was asked to audit the contracts for the Velar project.

During this audit we found one high issue, one medium issue and several minor issues. Also, several enhancements were proposed.

One minor issue was resolved, one high issue was partially resolved and the other issues were acknowledged. Two enhancements were implemented.

Scope

The audited files are from two different git repositories, which correspond to the same code but in different iterations.

The dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

First Iteration

Repository: https://github.com/Velar-co/mainnet .

Commit: `20708903c01bff64b1b4c23920bfd1b79a72d876`.

The scope for this iteration includes and is limited to the following files:

- `contracts/ft-plus-trait.clar`: SIP-10 trait with mint and burn functions.
- `contracts/univ2-core.clar`: UniswapV2-like core and factory contract.
- `contracts/univ2-fee-to-trait.clar`: Trait with the send-revenue function.
- `contracts/univ2-fee-to.clar`: Funds-accumulation contract.
- `contracts/univ2-library.clar`: Utility functions to calculate swap amounts.
- `contracts/univ2-router.clar`: UniswapV2-like router.
- `contracts/wstx-xusd.clar`: SIP010 token named wstx-xusd, with mint and burn.
- `contracts/wstx.clar`: SIP010 interface for STX.

It must be noted that the `contracts/wstx-xbtc.clar` file, present in the audited version but not referenced in the `Clarinet.toml` file, was not included in this audit.

Second Iteration

Repository: https://github.com/Velar-co/velar-1.0 .

Commit: `be59a79375bd8fa4aeccf498d223005ccda6c89d`.

The scope for this iteration includes and is limited to the following files:

- `staking-core/staking-core.clar`: Contract for stacking and unstacking Velar.
- `staking-periphery/staking-distributor.clar`: Contract for distributing Velar staking rewards.
- `farming/farming-send.clar`: Contract for tokens transferring and notifying the receiver.
- `farming/farming-receive-trait.clar`: Trait for contracts which are notified on token transfer.
- `farming/farming-wstx-velar-core.clar:` Contract for stacking and unstacking wSTX-Velar LP token.
- `farming/farming-wstx-velar-distributor.clar:` Contract for distributing wSTX-Velar farming rewards.
- `univ2-lptokens/template.clar`: Template for LP token.
- `univ2-lptokens/wstx-sbtc.clar`: wSTX-sBTC LP token.
- `univ2-lptokens/wstx-velar.clar`: wSTX-Velar LP token.
- `univ2-lptokens/wstx-xbtc.clar`: wSTX-xBTC LP token.
- `tokens/neebs.clar`: Neebs token.
- `tokens/velar.clar`: Velar token.
- `util/util-multisend.clar`: Utility for transferring to many recipients at once.

And this is a map from first iteration scope to second iteration scope, where contracts were renamed or relocated:

- `contracts/ft-plus-trait.clar` -> `contracts/univ2-core/ft-plus-trait.clar`
- `contracts/univ2-core.clar` -> `contracts/univ2-core/univ2-core.clar`
- `contracts/univ2-fee-to-trait.clar` -> `contracts/univ2-core/univ2-share-fee-to-trait.clar`
- `contracts/univ2-fee-to.clar` -> `contracts/univ2-core/univ2-share-fee-to.clar`
- `contracts/univ2-library.clar` -> `contracts/univ2-periphery/univ2-library.clar`
- `contracts/univ2-router.clar` -> `contracts/univ2-periphery/univ2-router.clar`
- `contracts/wstx-xusd.clar` -> `contracts/univ2-lptokens/wstx-xusd.clar`
- `contracts/wstx.clar` -> `contracts/tokens/wstx.clar`

Methodology

CoinFabrik was provided with the source code. Our auditors spent one week 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. Without being limited to them, the audit process included the following analyses.

- Arithmetic errors
- Race conditions
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

ID	Title	Severity	Status
HI-01	Authentication via tx-sender	High	Partially resolved
ME-01	Block Time Assumption Broken on Nakamoto Release	Medium	Acknowledged
MI-01	Convoluted Fees	Minor	Acknowledged
MI-02	check-fee Rounding Errors	Minor	Resolved

ID	Title	Severity	Status
MI-03	Panicking on Possible Error	Minor	Acknowledged

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. They must be fixed **immediately**.
- **High:** These refer to a vulnerability that, if exploited, could have a substantial impact, but requires a more extensive setup or effort compared to critical issues. These pose a significant risk and **demand immediate attention**.
- **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 in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit has one of the following statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision. The reported risk is accepted by the development team.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

Critical Severity Issues

No issues found.

High Severity Issues

HI-01 Authentication via tx-sender

Location:

- `contracts/univ2-core.clar: 32,41,265,283,358,425,566`
- `contracts/univ2-fee-to.clar: 18,27`
- `contracts/wstx-xusd.clar: 18,46`
- `tokens/neebs.clar: 15, 29`
- `tokens/velar.clar: 15, 29`
- `contracts/staking-periphery/staking-distributor.clar: 17`
- `contracts/farming/farming-wstx-velar-distributor.clar: 22`

The system utilizes `tx-sender` for its authentication processes. This method, while functional, presents latent vulnerabilities, particularly exposing actors within the system to threats known as phishing[1].

Actors could inadvertently activate a malicious contract. Once activated, the deceptive contract can access and initiate certain functions, presenting actions as if they were done by the original actor. This impersonation potential poses risks, depending on the specific function being accessed.

In particular, all the actions made by all the roles described for all the contracts in the Privileged Roles can be a target for a phishing attack.

Recommendation

It is advisable to switch from using `tx-sender` to `contract-caller` for a more reliable and secure authentication method. It must be noted that when `tx-sender` is used as part of an `as-contract` invocation it does not lead to this issue, as it evaluates to the contract's principal. Introducing a white list for trusted callers can add an extra layer of security, particularly if the system needs to interact with specific intermediary contracts. These intermediate contracts should properly check their `contract-caller` and/or pass it to the contract where the check needs to be made.

Status

Partially resolved. Contract-caller implemented only in the following lines:

- `contracts/univ2-core.clar: 32`

[1] https://www.coinfabrik.com/blog/tx-sender-in-clarity-smart-contracts/

For the rest of the instances, the development team decided to keep `tx-sender` for compatibility with existing tokens. In that case, the issue is mitigated by proper use of post-conditions.

Medium Severity Issues

ME-01 Block Time Assumption Broken on Nakamoto Release

Location:

- `contracts/farming/farming-wstx-velar-core.clar: 19`
- `contracts/staking-core/staking-core.clar: 19`

Farming and stacking core contracts assume block time for the calculation of epoch lengths. However, this assumption is expected to be modified in the next Stacks upgrade (Nakamoto Release), which will reduce block time.

Recommendation

Instead of relying on Stacks block time, rely on Bitcoin block time which is not expected to change. For this, replace `block-height` instances for `burn-block-height`.

Status

Acknowledged. The development team decided to keep this assumption.

Minor Severity Issues

MI-01 Convoluted Fees

Location:

- `contracts/univ2-core.clar:528-548`

The calculation of the fees for swapping in `contracts/univ2-core.clar` is non-intuitive

1. The swap fee is really the amount that is not fees.
2. The protocol fee is calculated against the fee total, and not the total value.
3. The share fee is calculated against the protocol fee, and not the total value nor the fee amount.

This convoluted way to calculate fees may cause misunderstanding in the users on how the fees are charged.

It must also be noted that the fees are calculated in the `calc-swap` function. This name is also misleading, as it implies that the obtained token amount is calculated instead.

Recommendation

Calculate all the fees directly based on the in amount. Choose proper names to describe fees.

Status

Acknowledged. The development team decided to keep the current fee system.

MI-02 check-fee Rounding Errors

Location:

- `contracts/univ2-core.clar: 122-129`

In the `check-fee` function of the `contracts/univ2-core.clar` contract, if the denominator of the either fee or guard does not divide 1000000, there may be rounding errors leading to accepting a slightly bigger fee than guard.

Recommendation

Choose a single denominator for all the fees. Compare only the fee numerators.

Status

Resolved. Fixed according to the recommendation.

MI-03 Panicking on Possible Error

Location:

- `contracts/staking-periphery/staking-distributor.clar: 161, 226`
- `contracts/farming/farming-wstx-velar-distributor.clar: 125, 154`

Using `unwrap-panic` results in the transaction being finished because of a runtime error when the provided value is an error or a none. The runtime error does not allow the caller to handle that error and act in response. Also, this kind of error does not provide any information about the reason for the reverted transaction to the user.

While that form is a convenient method to unwrap values, it should not be used unless it is impossible to trigger the panic.

Recommendation

Replace `unwrap-panic` for `unwrap!` when there is a flow which might trigger an error or none value.

Status

Acknowledged. The development team decided not to change the error handling.

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

ID	Title	Status
EN-01	Proper Project	Implemented
EN-02	Commented Code	Implemented
EN-03	Unused Data in Blockchain	Not implemented

EN-01 Proper Project

The provided source code does not pass the `clarinet check` command. When ran in the command line it fails:

```
$ clarinet check
error: unable to read file /audits/Velar-co-mainnet/settings/Devnet.toml
Os { code: 2, kind: NotFound, message: "No such file or directory" }
```

Also no tests for the clarinet test command were provided

Recommendation

Provide a proper project to assess and test the contracts.

Status

Implemented.

EN-02 Commented Code

Location:

- `contracts/univ2-library.clar: 90-133`

Recommendation

Remove the commented code.

Status

Implemented.

EN-03 Unused Data in Blockchain

Location:
- `contracts/univ2-core.clar: 70-79`

The `symbol`, `block-height` and `burn-block-height` fields in the pool map of the `contracts/univ2-core.clar` contract are not needed for the contract functionality.

Recommendation

Remove the unneeded data to save on transaction fees and simplify the code.

Status

Not implemented.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

Centralization

The owner of the `contracts/univ2-core.clar` contract can set the fees and the where the fees go to.

The owner of the `contracts/univ2-fee-to.clar` contract can withdraw funds from the contract.

The owner of the `contracts/wstx-xusd.clar` contract can mint and burn tokens that belong to any user. However, the owner is hardcoded to `univ2-core` contract, and this only burns tokens from the caller.

For the second iteration, token contracts also have an owner who can mint and burn them, even if they belong to a user, but with the owner hardcoded to `univ2-core`. Also, `contracts/staking-periphery/staking-distributor.clar` has an owner who can call the `receive` function, which is a post-transfer hook for updating contract balances.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

contracts/univ2-core.clar

Owner

The principal with the owner role can:

- set a new owner via the `set-owner` function.
- set the principal where fees are sent via `the set-fee-to` function.
- set the principal where the revenue is shared while swapping via the `set-rev-share` function.
- set a new swap fee for a pool via the `update-swap-fee` function.
- set a new protocol fee for a pool via the `update-protocol-fee` function.
- set a new share fee for a pool via the `update-share-fee` function.

The initial owner of the contract is the deployer.

Fee to

The principal with the "fee to" role can:

- collect accrued fees via the `collect` function.

The "fee-to" of the contract is the deployer, but given that standard principals cannot implement traits this functionality is not available until a new "fee-to" is set by the owner via the `set-fee-to` function.

contracts/univ2-fee-to.clar

Owner

The principal with the owner role can:

- set a new owner via the `set-owner` function.
- withdraw any SIP010 tokens owned by the contract via the `harvest` function.

The initial owner of the contract is the deployer.

contracts/wstx-xusd.clar

Owner

A principal with the owner role can:

- set the changeable owner via the `set-owner` function.
- mint wstx-xusd tokens for any principal via the `mint` function.
- burn wstx-xusd tokens for any principal via the burn function.

By default there are two owners. One is the `.univ2-core` contract, implemented in the `contracts/univ2-core.clar` file according to the `Clarinet.toml` file. The other can be changed and by default is the deployer of the contract.

Changelog

- 2024-01-10 – Initial report based on commit `20708903c01bff64b1b4c23920bfd1b79a72d876`.
- 2024-02-09 – Second iteration based on commit `be59a79375bd8fa4aeccf498d223005ccda6c89d`. Fixes on first iteration findings were checked. HI-01 was updated with new locations for this issue.
- 2024-02-22 – Final report updating status for the findings from the second iteration.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the Velar project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.