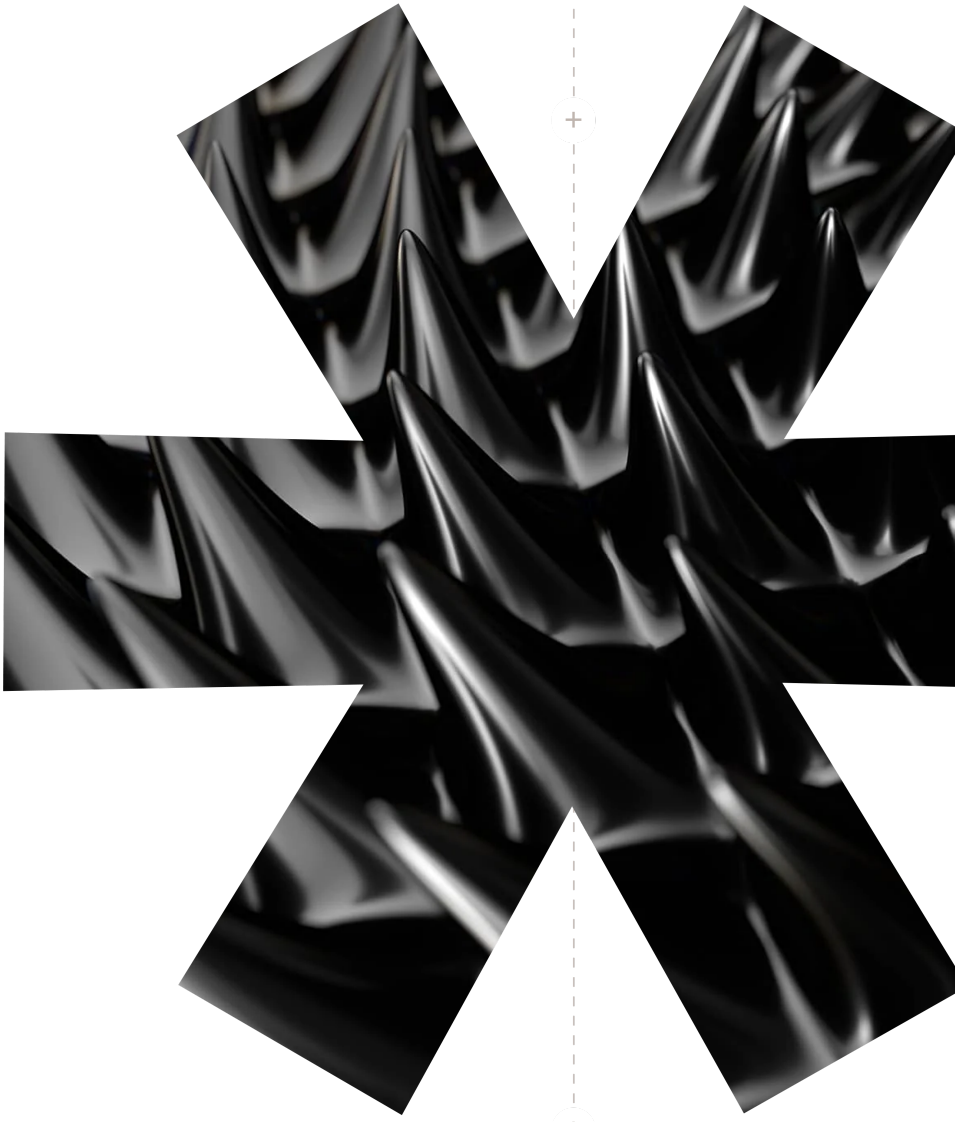
thesisdefense

Security Audit Report

Velar Perpdex

Clarity Smart Contracts

Initial Report // May 23, 2024
Final Report // July 11, 2024



Team Members

Jehad Baeth // Senior Security Auditor
Justin Regele // Senior Security Auditor
Mukesh Jaiswal // Senior Security Auditor
Bashir Abu-Amr // Head of Delivery

Table of Contents

About Thesis Defense

Thesis Defense serves as the auditing services arm within Thesis, Inc., the venture studio behind tBTC, Fold, Taho, Etcher, and Mezo. Our team of security auditors have carried out hundreds of security audits for decentralized systems across a number of technologies including smart contracts, wallets and browser extensions, bridges, node implementations, cryptographic protocols, and dApps. We offer our services within a variety of ecosystems including Bitcoin, Ethereum + EVMs, Stacks, Cosmos / Cosmos SDK, NEAR and more.

Thesis Defense will employ the Thesis Defense Audit Approach and Audit Process to the in scope service. In the event that certain processes and methodologies are not applicable to the in scope services, we will indicate as such in individual audit or design review SOWs. In addition, Thesis Defense provides clear guidance on successful Security Audit Preparation.

`Section_1.0`

Scope

Technical Scope

- **Repository:** https://github.com/Velar-co/gl-v0.1
- **Audit Commit:** 2a9e943757c90dc8e61ec5ca20a7c9a084935b4f
- **Verification Commit:** ff8c8b781bbab687a91ff5d0779b451580b5b62f
- **Files in Scope:**
    - gl-core.clar
    - gl-fees.clar
    - gl-math.clar
    - gl-pools.clar
    - gl-positions.clar
    - gl-oracle.clar
- **Files Out of Scope:**
    - The `liquidate` function in `gl-positions.clar`

Documentation

- None provided.

Section_2.0

Executive Summary

Schedule

This security audit was conducted from May 6, 2024 to May 23, 2024 by 3 security auditors for a total of 5 person-weeks and 3 person-days.

Overview

This report details the findings and outcomes of our security audit of Velar Clarity Smart Contracts for Velar Perpdex. The Velar Clarity Smart Contracts are a series of smart contracts written in Clarity for the Stacks ecosystem. Velar Perpdex provides an external API for traders to trade using short and long positions against stable (quote) tokens using the base token of the chain.

Shortly before the start of the audit, the Stacks Nakamoto delay was announced. This change impacted the codebase in numerous ways, and made the Velar implementation a moving target.

We found several critical and high severity issues, including across the design and implementation. If the project were launched today, these issues would be catastrophic. It's clear, however, that most issues are related to the change in development timeline. We're confident the team will address these issues before a production release.

Threat Model

Prior to auditing the code, we determined relevant areas of concern and attack vectors to guide our auditing investigation:

- An attacker front-runs another transaction and cause the trader lose value;
- An attacker front-runs a liquidation, but intentionally cause the liquidation to fail by using TX errors to adversely affect the liquidity pools (this is explicitly out of scope);
- An attacker can close a position they did not open;
- A trader can close a position twice (get 2x rewards);
- A trader can close a position without penalty (in case of loss);
- A trader can leverage non-existent or incorrect tokens;
- A trader can burn another traders tokens;
- A trader can mint more value than deposited;
- A trader can trigger an overflow condition by setting the number of decimals to a value higher than Clarity can handle, and therefore perpetually break a certain portion, or the entirety of, the protocol;
- Contracts don't shift tx-sender context which leads to undefined or dangerous behavior;
- Unlocking more reserves than intended by getting the base/quote-interest to 0;
- An attacker leverages fee transfers between short and long positions in malicious ways;
- An attackers leverages static fee calculations to take control and deplete liquidity pools;
- An attacker creates a malicious contract that proxies calls to create pool, and tricks the OWNER of the protocol into making calls to the proxy contract; and
- An attacker manipulates the price oracle to deplete the liquidity pool.

Security Best Practices Assessment

We performed an assessment of the overall security posture of the Velar Perpdex, including identifying strengths and areas for improvement in the system design and implementation, in addition to feedback on the project documentation, test suite, and the use of dependencies.

Security by Design

We conducted a thorough design review to confirm adherence to decentralized system design best practices, and the absence of common design issues that could result in security vulnerabilities.

The Oracle

The security of the Velar protocol is largely dependent on the security of the oracle which determines asset prices. If the oracle functionality is compromised, trader's can set their own price at opening or closing of positions, which can result in withdrawal of an assets amount that is greater than the trader's collateral is worth.

We found critical issues in the design and implementation of the oracle and have documented these findings in this report. However, through our discussions with the Velar team, we understand that the design of the oracle for the Velar protocol is a work in progress. Given we can only audit what is in scope, we have addressed the oracle issues as currently implemented, and hope our feedback can inform and contribute to shaping a more secure design moving forward. Once a final design for the oracle is decided upon and implemented, we recommend that the Velar team will have that design and implementation audited again.

In the current implementation, the spot asset price is fetched from the oracle and submitted, along with a signed hash of the current spot price, with each trader request to mint, burn, open, or close. The current spot price is saved for each block. If no price has been set for the current block, then the signed current spot price is set for the block. This means that each block has a static price for the block, no matter how long it takes for the block to finish.

The signed oracle message also does not include a nonce, which makes the signatures vulnerable to replay attacks (Issue 1). We also discovered that while the signature is verified, the verification itself is disabled (Issue 2). We recommend enabling the check.

Use of `block-height`

The protocol uses `block-height` as a timestamp in determining asset prices for a given block. The result is that the price of the asset can only be updated once per block. This will result in the price not being responsive to rapid price movements, and that the returned price diverging significantly from the actual price of the asset. We recommend storing price by timestamp instead of `block-height` (Issue 10).

We also found that the current implementation uses block-height. This variable will become obsolete once the Nakamoto upgrade goes into effect. As a result, we recommend using `stacks-block-height` in anticipation of the Nakamoto upgrade (Issue 13).

Cross Contract Forgery

One of the intended uses of the Velar protocol is for traders to perform trades using contracts, and that these contracts will need to perform operations on behalf of the traders' externally owned accounts. However, this feature of Clarity can also be weaponized by attackers to perform Cross-Contract Request Forgery. We make recommendations on how to protect against this without losing the intended functionality (Issue 8).

We investigated whether attackers could manipulate closing positions, either by closing other users positions, closing positions (and reaping rewards) twice, or closing positions without penalty in the case of loss. We found that when opening and closing positions, the `tx-sender` is always used for defining the user account, and that the user account is never a user supplied variable.

The preconditions of the close function call also assert that the `tx-sender` user matches the user of the position. While this does not allow other accounts to close a user's positions, this construction is still vulnerable to Cross-Contract Request Forgery attacks, where an attacker can trick a user into performing an action on a malicious contract that makes a new request to the `gl-core` contract to perform actions on `tx-sender` 's behalf.

We recommend implementing a new function where traders can submit a list of contracts they authorize to perform trades on their behalf. This new function must not allow any intermediary contracts to call it, as that would only move the vulnerability to a different place rather than remediate it. Instead, the user is responsible for managing their own list of authorized contracts, and this list is consulted whenever `contract-caller?` does not equal `tx-sender` (Issue 8).

Design Best Practice

We found that that the protocol does not implement an emergency pause mechanism (Issue 7). This prevents the owner of the protocol from taking action in case of an ongoing attack.

Furthermore, the protocol does not implement a multi-step process for transferring ownership or the fee collector role which could result in irrevocable loss of control of these security critical roles (Issue 12).

Secure Implementation

We conducted an in-depth examination and manual review of the project's source code to assessed the correctness of the implementation, compliance with best practices, and adherence to design specifications. We identified security vulnerabilities and code quality issues.

We found that attackers cannot close a position more than once, and that penalties were unavoidable for trades that lost funds. However, we did discover that the updated local state was not being returned when a position was closed (Issue 16). This could affect the test suite by hiding code regression by testing against the incorrect data.

We also identified that a dishonest trader could create many low collateral positions across multiple addresses and bypass collateral fees (Issue 6). There were also some implementation errors related to the calculation of total supply of the pool and tokens (Issues 3, 5, 14). We also found edge cases where runtime errors would prevent user's from trading in positions that would be advantageous to them (Issues 4 and 17).

Project Documentation and Tests

For this security audit, our team was not provided with any project documentation. This inhibited our audit as we did not have a reference of how the protocol was intended to work. We recommend the development of project documentation that accurately reflects that state and intended functionality of the implementation.

We also found few code comments describing security critical components and functions in the protocol. We recommend writing comprehensive code comments.

There were tests for each of the contracts, as well as some test scenarios. However, we did not find the tests to be comprehensive. We recommend implementing a comprehensive test suite that includes unit and end to end tests.

Use of Dependencies

The protocol relies on an external oracle for determining asset prices. We identified issues in the use of this dependency (see The Oracle).

Section_3.0

Key Findings Table

Issues	Severity	Status
ISSUE #1 Oracle Signature is Vulnerable to Replay Attacks	Critical	Not Fixed
ISSUE #2 Oracle Signature Check is Disabled	Critical	Not Fixed
ISSUE #3 The `price` Function Will Revert if the `current` Price and `desired` Price are Equal	High	Fixed
ISSUE #4 Runtime Error Prevents Closing Positions When Asset Price Drops to Zero	High	Partial
ISSUE #5 Incorrect Calculation of LP Tokens	High	Fixed
ISSUE #6 Incorrect Collection of Protocol Collateral Fees	High	Not Fixed
ISSUE #7 Protocol Does Not Have an Emergency Pause Mechanism	Medium	Not Fixed
ISSUE #8 Cross-Contract Request Forgery	Medium	Not Fixed
ISSUE #9 Oracle Verification Key Cannot Be Rotated	Medium	Not Fixed
ISSUE #10 Asset Price Should be Set by `timestamp` and not `block-height`	Medium	Not Fixed
ISSUE #11 The `close` Function Returns a Premature State of the `position` Object	Medium	Not Fixed
ISSUE #12 Protocol Owner and Fee Collector are not Updated in Multi-Step Operation	Medium	Not Fixed
ISSUE #13 Clarity `block-height` Will Become Obsolete After the Nakamoto Upgrade	Medium	Not Fixed
ISSUE #14 Check for Total Supply is Performed After Calculating the LP Amount	Low	Fixed
ISSUE #15 The `INVARIANT` Function is Not Used in the `mint` Function	Low	Not Fixed

Issues	Severity	Status
ISSUE #16 Incorrect Local State Update	Low	Fixed
ISSUE #17 Closing a Position in the Same Block It was Opened in Produces a Runtime Error	Low	Not Fixed
ISSUE #18 Arbitrary Amount for Margin Protection	Low	Not Fixed
ISSUE #19 The 'protocol' Variable is Unused	None	Fixed
ISSUE #20 Memo Field Omission in `ft-transfer?` Operations	None	Not Fixed
ISSUE #21 Improper Use of Assertions in `gl-pools` Contract	None	Not Fixed

Severity definitions can be found in Appendix A

Section_4.0

Findings

We describe the security issues identified during the security audit, along with their potential impact. We also note areas for improvement and optimizations in accordance with best practices. This includes recommendations to mitigate or remediate the issues we identify, in addition to their status before and after the fix verification.

ISSUE#1

Oracle Signature is Vulnerable to Replay Attacks

Critical Not Fixed

Location

gl-oracle.clar#L44-L46

Description

The price oracle verification process is vulnerable to replay attacks. Because no nonce is used when signing the oracle output, attackers can find a signed price that is desirable and include this price and signature in a request.

Impact

Attackers can manipulate asset prices when opening or closing a position to drains assets from the protocol.

Recommendation

We recommend including a nonce in the signed oracle price output and ensure the nonce never repeats.

Verification Status

Velar Perpdex team stated that the issue has not been addressed because the Oracle architecture and code will change significantly once better solutions are available in the Stacks ecosystem.

We noted that the current oracle implementation is used for testing purposes.

ISSUE#2

Oracle Signature Check is Disabled

Critical Not Fixed

Location

gl-oracle.clar#L61

Description

The assertion statement that guarantees that the oracle price signature is correct is commented out in the code. This disables the signature verification check.

Impact

Attackers could set arbitrary prices when opening and closing positions allowing the attacker to open a position at a very low price, and close it a very high price. Asset price manipulation can lead to draining of protocol assets.

Recommendation

We recommend creating a set of functions where the owner of the protocol can update the verification key for the oracle.

Verification Status

Velar Perpdex team stated that the issue has not been addressed because the Oracle architecture and code will change significantly once better solutions are available in the Stacks ecosystem.

We noted that the current oracle implementation is used for testing purposes.

ISSUE#3

The `price` Function Will Revert if the `current` Price and `desired` Price are Equal

High Fixed

Location

gl-oracle.clar#L70

Description

Two edge cases in the `check-slippage` function will cause the function to return false. This will cause the `asserts! acceptable` call in the `price` function to throw an error. Under these edge cases user's will be unable to open, mint, close, or burn when the `current` price is their `desired` price.

The `check-slippage` function first checks if `current` is greater than `desired`, and then subtracts the lesser from the greater and returns true if that value is less than the slippage. Because slippage conceptually is the difference between `current` and `desired` price, when the calculated slippage equals exactly the slippage provided to the function, the function will return false because the comparison is non-inclusive (i.e. the comparison uses the less than <, rather than the less than or equal <= operator).

For example, the function is called with `current` price of 20, `desired` price of 10, and a slippage of 10. The function will check if `20-10<10` and because the result of `10<10` is not true, the condition will not be met, causing the function to return false which will cause the price function to revert.

A second case is where the current and desired price are the same, resulting in the difference being zero. If the slippage passed to the `check-slippage` function is calculated correctly as 0 (since slippage is the difference between current and desired price), then `check-slippage` will return false because 0 is not less than 0. This also will cause the `price` function to revert.

Impact

User's will be unable to open, mint, close, or burn when the `current` price is their `desired` price.

Recommendation

We recommend including the slippage when comparing it with the price, and ensure the slippage is restricted to the maximum allowable range. This could look like the following:

```
(<= (- desired current) slippage)
```

ISSUE#4

Runtime Error Prevents Closing Positions When Asset Price Drops to Zero

High Partial

Location

gl-math.clar#L53-L59

Description

When an asset price drops to zero, a runtime error occurs in `from-amount` in the `gl-math` contract due to a division by zero which prevents traders from closing positions.

When running unit tests with a scenario where the price of an asset is zero, the following runtime error message is produced.

```
 FAIL  tests/scenario/one.test.ts > overlapping long positions receiving fees Unknown
Error: Call contract function error: gl-core::close(u1,
'ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VGZJSRTPGZGM.velar,
'ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VGZJSRTPGZGM.wstx, u1, { current: 0x0000000000000000,
desired: u0, signature:
0x2a05ed525b8ddb3da40084090eb067e06292b707c1929dfa2847b297b3ab4e472dc9c7b3aeeb49bf9a97
slippage: u1 }) -> Error calling contract function: Runtime error while interpreting
ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VGZJSRTPGZGM.gl-core: Runtime(DivisionByZero,
Some([FunctionIdentifier { identifier: "ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VGZJSRTPGZGM.gl-
core:close" }, FunctionIdentifier { identifier: "_native_:special_let" },
...
FunctionIdentifier { identifier: "ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VGZJSRTPGZGM.gl-
math:from-amount" }, FunctionIdentifier { identifier: "_native_:native_div" }]))
```

Impact

Long positions should be automatically liquidated when their value drops to zero. The collateral for these positions would typically be used to cover the losses. However, due to a `DivisionByZero` error, these long positions will not be closed properly.

Recommendation

When closing the long position while price is zero, payout will be zero for the trader. We recommend not calling `quote-to-base` function to convert the final amount which will prevent a `DivisionByZero` error.

Verification Status

The `check-price` function in `gl-oracle.clar` ensures that the price is always greater than zero, so that price oracle can never return a price of zero or set the price to zero because the call will revert. The revert will prevent the price ever being set to zero for a block. This could result in the system becoming inoperable if the oracle price drops to total zero and does not move. Velar Perpdex team has stated this scenario is highly unlikely and is not anticipating such a scenario. This fix could also become obsolete when oracle code is changed to a new architecture.

ISSUE#5

Incorrect Calculation of LP Tokens

High Fixed

Location

gl-pools.clar#L210

Description

The LP amount is calculated incorrectly. Currently the input value (mint value) is divided by the sum of the total asset (pool value) plus the input (mint value). This prevents a division by zero error, which we suspect is the reason this calculation occurs in this fashion. However, this will dilute the amount of value that the liquidity provider provides to the pool:

`mv` : mint value `pv` : pool value

`f_0 : mv*total-supply / mv+pv` `f_1: mv *f_0 / mv+pv`

The correct calculation would calculate the LP amount by multiplying the minting value by the total supply, and dividing this by the pool value:

```
lp-amount = mv * total-supply / pv
```

When the pool value is initialized, this will always fail with a division by zero error. This error condition must be handled as part of initializing the liquidity pool with a special check (Issue 14).

Impact

Liquidity provider will be receiving the less LP tokens than expected.

Recommendation

We recommend correcting the LP token calculation to:

```
lp-amount = mv * total-supply / pv
```

ISSUE#6

Incorrect Collection of Protocol Collateral Fees

High Not Fixed

Location

gl-params.clar#L156

Description

If the trader supplied collateral is less than 1000, the collateral fee will always be zero. For amounts greater than 1000, the deducted fees will be lower than expected due to precision loss.

The protocol has no way of knowing if different EOA addresses are controlled by the same trader. This means that a malicious trader can make 10 positions with a collateral of 999 for each position and not pay any fees. These “no-fees” positions would be equivalent to an honest trader’s position with a collateral of 9990, but without fees. This will affect the liquidity pool by depriving it of trading fees.

In the case where collateral is greater than 1000 but not divisible by 1000, there will be a precision loss, and the fees will be lower than expected.

Impact

The trader’s reduced fee payment will result in the `gl-fees` contract receiving less than anticipated.

Recommendation

We recommend correcting the fee calculation. For the “no-fees” positions, a minimum deposit should be enforced to prevent the fees being zero.

For the precision loss aspect of the issue, a higher decimal precision should be used. In Solidity, there are libraries that assist with this issue such as ABDKMath64x64. However, in Clarity we are not currently aware that such a library exists in the ecosystem.

Verification Status

Velar Perpdex team has stated the attack is not viable as the transaction fees would be greater than anything the attacker could save.

ISSUE#7

Protocol Does Not Have an Emergency Pause Mechanism

Medium | Not Fixed

Location

velar_clarity_smart_contracts

Description

The Velar protocol has no way of shutting off trading or preventing withdrawals in case a system critical error occurs. An emergency shutoff mechanism to pause trading or temporarily freeze pools provides a critical safeguard if a hack is in progress.

Impact

In the event there is a security breach or a system failure, there is no way to halt attackers from draining protocol assets.

Recommendation

Implement the ability to pause the platform that is callable only by the protocol owner.

Verification Status

Velar Perpdex team has stated they intend to prevent opening new positions in a later version of the smart contracts. A full pause of the system is considered by the Velar team to be an issue from a philosophical perspective, whereby the traders' trust in the system would be compromised if they knew the system can arbitrarily be shut down or paused while they have open positions.

While this viewpoint has merits in that it intends to safeguard traders from malicious platform operators, it does not address situations where an active attack from an external actor threatens their investment.

ISSUE#8

Cross-Contract Request Forgery

Medium | Not Fixed

Location

gl-core.clar#L160

gl-core.clar#L217

gl-core.clar#L277

gl-core.clar#L339

Description

The `mint`, `burn`, `open`, and `close` functions in the `gl-core` smart contract set the user variable based on the value of `tx-sender`. In Clarity, this value will refer to the originator of the call, regardless of any intermediary contracts unless the developers of the intermediary contracts explicitly change contexts. Because the affected functions rely solely on the originator of the contract to perform actions, attackers can create malicious contracts for use in phishing campaigns that will relay function calls to perform actions on `gl-core` that the trader did not intend.

This can be seen as a protocol feature being weaponized as an attack vector. Because traders often use contracts for their trading, the protocol needs a way to transparently allow contracts to operate on behalf of traders. However, in the current state of the application, we were able to create requests that arbitrarily closed positions.

Impact

A legitimate trader of the system could be tricked into performing actions they did not intend. This could result in a trader accidentally performing any action a trader would normally make, but without their knowledge. This could also lead to closing positions at disadvantageous points in time.

Recommendation

It is necessary for the system that traders be able to use contracts to perform actions on their behalf. Our recommendation does not prohibit the use of contracts. Rather, we suggest making a check whether `contract-caller` is different than `tx-sender`. If these are different, a secondary check can be made whether the calling contract is in a pre-approved list of contracts for the specific `tx-sender`.

A separate function should be added whereby a `tx-sender` can approve contracts authorized to relay transactions to the protocol. This function should also be protected against cross-contract request forgery by prohibiting calls from intermediary contracts. This way only the original sender can approve a contract, and they cannot be tricked by a phishing campaign into authorizing malicious contracts to relay functions on their behalf.

ISSUE#9

Oracle Verification Key Cannot Be Rotated

Medium | Not Fixed

Location

gl-oracle.clar#L23

Description

The public key used for verifying the integrity of the price oracle is hard coded into the application. The security and integrity of the system is reliant on the security of the price oracle, which makes gaining access to the signing key a prime target for attackers.

Impact

If the signing key were every compromised for any reason, with a hardcoded verification key in the contracts, it would be impossible to lock down the system again once the signing key was compromised.

Recommendation

We recommend creating a set of functions enabling the owner of the protocol to update the verification key for the oracle.

Verification Status

Velar Perpdex team stated that the issue has not been addressed because the Oracle architecture and code will change significantly once better solutions are available in the Stacks ecosystem.

We note that the current oracle implementation is used for testing purposes.

Velar Perpdex team also stated that there isaa planned fix once the Oracle model is updated which will likely involve a DAO multi-sig architecture for safe rotation of the verification key.

ISSUE#10

Asset Price Should be Set by `timestamp` and not `block-height`

Medium | Not Fixed

Location

gl-oracle.clar#L47-L49

Description

The asset price is retrieved and set only once per block. Depending on the time for blocks to be committed to chain, the price could drastically fluctuate, resulting in a price that users can access lacking in precision and being inconsistent with the actual price.

Impact

Traders cannot respond quickly enough to movements in the price. The speed at which they can respond is directly linked to the pace of blocks being committed to chain.

Recommendation

We recommend committing price to local storage based on `timestamp`, and not `block-height`, allowing multiple price updates in each block.

Verification Status

The Velar Team believes on chain transaction fees combined with the low block time after the Nakamoto upgrade will result in higher position holding times for traders. However, if empirical evidence indicates otherwise, they have stated they are open to making the recommended changes.

ISSUE#11

The `close` Function Returns a Premature State of the `position` Object

Medium | Not Fixed

Location

gl-core.clar#L374-L383

Description

The `close` function in `gl-core` returns the `position` variable without updating it to the closed state. This means the output of `close` includes the description of `position` prior to closing.

This could have significant ramifications for the test suite being accurate.

Impact

The data returned by the function is outdated and will be unusable by software interacting with the contract.

Recommendation

We recommend updating the `position` object before returning it so it reflects the actual state of the closed position, including its closing price and which block it was closed at.

We also recommend reviewing the test suite functions to ensure that tests are not reliant on the outdated data.

ISSUE#12

Protocol Owner and Fee Collector are not Updated in Multi-Step Operation

Medium | Not Fixed

Location

gl-core.clar#L29-L32

gl-fees-bank.clar#L13-L14

Description

The `set-owner` and `set-fee-collector` functions set these roles for the protocol instantly. Best practices for transferring ownership involves a two-step process where a new owner is first proposed, and is only set as the new owner when the new owner accepts control. This prevents the protocol from accidentally being transferred to an address that cannot control the protocol.

Impact

The ownership of the protocol could irrecoverably be transferred to an uncontrolled account.

Recommendation

We recommend implementing a two-step process for transferring administrative roles. Specifically, introduce a “confirm-role-transfer” function that requires explicit confirmation from the new owner of fee collector before the transfer is executed.

The updated implementation should ensure that the “set-admin” function only queues the transfer, and the actual transfer is only executed when the new owner or fee collector explicitly calls “confirm-role-transfer”. This added layer of validation helps prevent unintended consequences and enhances the overall security of the protocol.

Verification Status

The recommendation was not implemented at the time of this verification, however the Velar team stated that this fix is planned.

ISSUE#13

Clarity `block-height` Will Become Obsolete After the Nakamoto Upgrade

Medium | Not Fixed

Location

gl-oracle.clar#L47-L48

gl-positions.clar#L402-L403

gl-fees.clar#L210-L211

Description

During our review of the Velar Clarity smart contracts, we discovered several instances of the `block-height` variable being used across multiple contracts. Specifically, it is used for storing local state in the `gl-oracle` contract, position status in the `gl-positions` contract, and calculating fee deltas in the `gl-fees` contract.

The issue lies in the fact that `block-height` will become obsolete in Clarity v3, as stated in the Nakamoto release documentation. This change was introduced to ensure compatibility with Stacks' block height. Furthermore, `block-height` has a different meaning starting from the Nakamoto release of Stacks.

Impact

The use of `block-height` in these contracts may lead to unexpected behavior or errors when the Clarity v3 update is implemented, as it will no longer be supported. This could result in security vulnerabilities, data loss, or operational disruptions.

Recommendation

We recommend replacing all instances of `block-height` with `burn-block-height` or `stacks-block-height`, depending on the specific use case and requirements.

We also recommend reviewing and refactoring the contracts to ensure they are compatible with Clarity v3 and Stacks' block height. This may involve reevaluating the logic and assumptions used in these contracts.

Verification Status

The recommendation was not implemented at the time of this verification, however the Velar team stated that this fix is planned.

ISSUE#14

Check for Total Supply is Performed After Calculating the LP Amount

Low Fixed

Location

gl-pools.clar#L199

Description

The calculation of the amount of LP tokens depends on the base and quote tokens provided by liquidity providers, and the total `tokensupply` of the of LP token. The `calc-mint` function in `gl-pools` checks whether `totalsupply` is zero after calculating the LP token.

The calculation of the LP token is performed with the lines:

```
(f_0 (f mv pv total-supply)) (f_1 (f mv pv f_0))
```

The check for total supply is performed at the line:

```
(ok (if (is-eq total-supply u0) ;;assert pv == 0
```

This check for total supply should happen before the calculation of the LP token because if the total supply is zero, then the pool value will also be zero.

Impact

Currently, the function does not revert when the pool value is zero. However, once the LP token amount is calculated correctly Issue 5 it will prevent liquidity providers from supplying liquidity.

Recommendation

We recommend adding a zero check for the total supply before calculating the liquidity providers LP token amount.

ISSUE#15

The `INVARIANT` Function is Not Used in the `mint` Function

Low | Not Fixed

Location

gl-core.clar#L150

Description

The `INVARIANT` function has a safety check for the pool balance. When there is a change in the state of funds in the pool due to burn, open, or close operations, a check is performed to ensure that the pool balance exceeds the total value of positions held by both long and short holders. However, this check is implemented in the `burn`, `open`, and `close` functions, but not in the `mint` function.

Impact

Due to inconsistent checks, if the `mint` function lacks the same safety checks as the `burn`, `open`, and `close` functions, it could result in situations where the pool balance is insufficient to cover the total value of positions held by both long and short holders.

Recommendation

We recommend implementing safety checks consistently when LP providers provide liquidity.

Verification Status

The `INVARIANT` function has been added to the `mint` function but is currently commented out and disabled. While changes have been made to the code, they are not currently taking effect.

ISSUE#16

Incorrect Local State Update

Low | Fixed

Location

gl-positions.clar#L403-L404

Description

At the end of a successful position closure transaction, `exit-price` is not included in the update of the position, leaving the local state with incomplete data.

Impact

Local state does not include exit price information after a position is closed. This leaves local state out of sync with reality and does not provide other applications that interact with the protocol a proper view of the system.

Recommendation

We recommend including the `exit-price` in the `insert` and `merge` calls in `gl-positions` when the position is closed.

ISSUE#17

Closing a Position in the Same Block It was Opened in Produces a Runtime Error

Low | Not Fixed

Location

gl-fees.clar#L264

Description

Closing a position in the same block that the position was opened produces a runtime error in the `fees-at-block` function. This is because the function uses the current block height as input to the `get-block-info?` function. Because the block has not been mined yet, it returns `None` which causes the surrounding `unwrap-panic` call to fail.

Impact

This issue prevents opening and closing a position in the same block.

Recommendation

We recommend refactoring fee storage based on `timestamp` so that traders can open and close a position based on time and not on block height. This would also require that price is not stored per block, but also per `timestamp` Issue 10. If by design the system should not allow opening and closing positions in the same block, then there should be an assertion to prevent such a case and handles this edge case with an appropraite error.

Verification Status

Velar Perpdex team has stated that allowing closing a position in the same block as it is opened is intended by design.

ISSUE#18

Arbitrary Amount for Margin Protection

Low | Not Fixed

Location

gl-params.clar#L77-L79

Description

The `is-legal-position` function checks whether the leverage is less than 10x and whether this position value is less than base reserve or the quote reserve. However, while checking for the position value it multiplies the position value by an arbitrary amount:

```
(if (get long position)
   (>= (get base-reserves  pool) (* (get interest position) u50))
   (>= (get quote-reserves pool) (* (get interest position) u50)))
;; ...
```

When the interest position is increased by 50 times, the system checks whether the protocol can handle 50 such positions, but it will limit the user's ability to open a position.

For example, if the protocol has 1 million USD (base asset + quote asset), and a user with 30k USD wants to open a long position of 300,000 USD, the `open` function will check this condition and it will fail:

```
(>= (1,000,000) (1,500,000))
```

The protocol has the reserves to handle the position, but due to the above check, it fails.

Impact

The trader will not be able to open a position despite the availability of protocol liquidity to handle the position which is not an optimal use of protocol assets..

Recommendation

We recommend refactoring the check to `(>= (get base-reserves pool) (get interest position))` .

Verification Status

The recommendation was not implemented at the time of this verification, however the Velar team stated that this fix is planned.

ISSUE#19

The ‘protocol’ Variable is Unused

None Fixed

Location

gl-core.clar#L340

Description

The `protocol` variable is assigned but never used. This makes the code less readable and more confusing for future developers.

Impact

Informational – no security impact.

Recommendation

We recommend removing the assignment of the `protocol` variable.

ISSUE#20

Memo Field Omission in `ft-transfer?` Operations

None Not Fixed

Location

gl-core.clar#L50-L60

Description

The functions `call-transfer-from` and `call-transfer-to`, which make a call to the fungible token contracts, do not include the memo field in successful `ft-transfer?` operations. As per SIP010 recommendations, it is recommended to pass the nonce explaining the transfer operation.

In Stacks 2.0, the memo field will not be included in the event emitted by successful `ft-transfer?` operations. To ensure that the memo is emitted correctly, it is crucial to add a print statement if the `ft-transfer?` operation is successful and the memo is not `none`. The memo should be unwrapped and emitted after the `ft-transfer?` operation.

Impact

Informational – no security impact.

Recommendation

We recommend implementing a print statement that emits the memo after the successful `ft-transfer?` operation. This can be achieved by modifying the `call-transfer-from` and `call-transfer-to` functions to include the memo field.

ISSUE#21

Improper Use of Assertions in `gl-pools` Contract

None | Not Fixed

Location

gl-pools.clar#L343

Description

The `gl-pools` contract contains an assertion that is not being used correctly. Specifically, the line `(asserts! true err-close-preconditions)` is not performing any actual checks or actions, and is likely intended to be replaced with an appropriate error handling mechanism.

Impact

Informational – no security impact.

Recommendation

To address this issue, we recommend the following steps:

1. Remove the redundant assertion from `gl-pools` and replace it with a proper error handling mechanism.
2. Update the similar pattern in `gl-core` to use `unwrap!` correctly, providing a meaningful error message. For Example:

```
(unwrap! (contract-call? .gl-pools close id deltas) err-close-postconditions)
```

Section_5.0

Appendix A

Severity Rating Definitions

At Thesis Defense, we utilize the Immunefi Vulnerability Severity Classification System - v2.3.

Severity	Definition
Critical	• Manipulation of governance voting result deviating from voted outcome and resulting in a direct change from intended effect of original results • Direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield • Direct theft of any user NFTs, whether at-rest or in-motion, other than unclaimed royalties • Permanent freezing of funds • Predictable or manipulable RNG that results in abuse of the principal or NFT • Protocol insolvency
High	• Theft of unclaimed yield • Theft of unclaimed royalties • Permanent freezing of unclaimed yield • Permanent freezing of unclaimed royalties • Temporary freezing of funds • Temporary freezing NFTs
Medium	• Smart contract unable to operate due to lack of token funds • Enabling/disabling notifications • Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol) • Theft of gas • Unbounded gas consumption
Low	• Contract fails to deliver promised returns, but doesn't lose value
None	• We make note of issues of no severity that reflect best practice recommendations or opportunities for optimization, including, but not limited to, gas optimization, the divergence from standard coding practices, code readability issues, the incorrect use of dependencies, insufficient test coverage, or the absence of documentation or code comments.

Section_6.0

Appendix B

Thesis Defense Disclaimer

Thesis Defense conducts its security audits and other services provided based on agreed-upon and specific scopes of work (SOWs) with our Customers. The analysis provided in our reports is based solely on the information available and the state of the systems at the time of review. While Thesis Defense strives to provide thorough and accurate analysis, our reports do not constitute a guarantee of the project's security and should not be interpreted as assurances of error-free or risk-free project operations. It is imperative to acknowledge that all technological evaluations are inherently subject to risks and uncertainties due to the emergent nature of cryptographic technologies.

Our reports are not intended to be utilized as financial, investment, legal, tax, or regulatory advice, nor should they be perceived as an endorsement of any particular technology or project. No third party should rely on these reports for the purpose of making investment decisions or consider them as a guarantee of project security.

Links to external websites and references to third-party information within our reports are provided solely for the user's convenience. Thesis Defense does not control, endorse, or assume responsibility for the content or privacy practices of any linked external sites. Users should exercise caution and independently verify any information obtained from third-party sources.

The contents of our reports, including methodologies, data analysis, and conclusions, are the proprietary intellectual property of Thesis Defense and are provided exclusively for the specified use of our Customers. Unauthorized disclosure, reproduction, or distribution of this material is strictly prohibited unless explicitly authorized by Thesis Defense. Thesis Defense does not assume any obligation to update the information contained within our reports post-publication, nor do we owe a duty to any third party by virtue of making these analyses available.