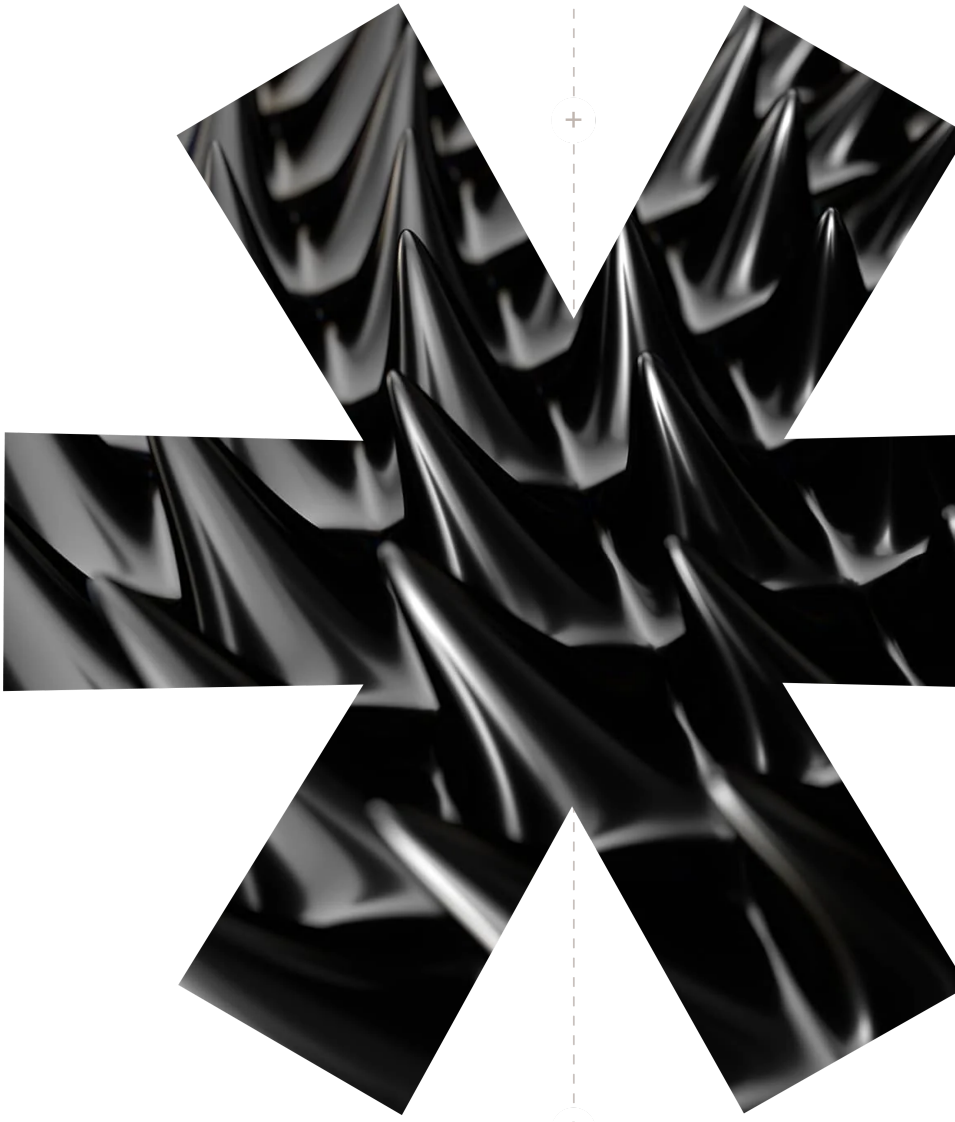
thesisdefense

Security Audit Report

Velar Perpdex

Vyper Smart Contracts

Initial Report // June 12, 2024
Final Report // July 09, 2024



Team Members

Ahmad Jawid Jamiulahmadi // Senior Security Auditor
Mukesh Jaiswal // Senior Security Auditor
Bashir Abu-Amr // Head of Delivery

Table of Contents

1.0 Scope — 4

- 1.1 Technical Scope
- 1.2 Documentation

2.0 Executive Summary — 5

- 2.1 Schedule
- 2.2 Overview
- 2.3 Threat Model
- 2.4 Security By Design
- 2.5 Secure Implementation
- 2.6 Project Documentation & Tests
- 2.7 Use of Dependencies

3.0 Key Findings Table — 7

4.0 Findings — 9

- 4.1 Total Supply is Not Decreased While Burning Tokens
  Critical Fixed
- 4.2 Opening and Closing a Position in the Same Block Can Cause a Pool Imbalance
  High Fixed
- 4.3 Edge Cases While Checking Slippage Will Cause `price` Function to Revert (Known Issue)
  High Fixed
- 4.4 Long Positions Remain Open Even If the Asset Price Falls to Zero (Known Issue)
  High Not Fixed
- 4.5 Issues With Precision Loss
  High Not Fixed
- 4.6 Only One Operation Per Block Is Allowed Due to Oracle Price Check
  High Fixed
- 4.7 Protocol Fees Cannot Be Withdrawn (Known Issue)
  High Not Fixed
- 4.8 The Protocol Locks Reserves Indefinitely
  High Not Fixed
- 4.9 `calc_pnl_long` Function Can Prevent Closure of Valid Positions and Deprive the User From Receiving Profit
  High Partial
- 4.10 Incorrect Calculation of Positions Fees
  High Not Fixed
- 4.11 Protocol Owner is Not updated in Multi Step Operation (Known Issue)
  Medium Not Fixed
- 4.12 Protocol Does Not Have an Emergency Pause Mechanism (Known Issue)
  Medium Not Fixed
- 4.13 Unchecked Return Value From Token Transfer
  Medium Fixed
- 4.14 Protocol Does Not Handle Fee on Transfer Tokens
  Medium Not Fixed
- 4.15 Tokens Insufficiently Validated
  Medium Not Fixed

↳ 4.16 Arbitrary Amount for Margin Protection (Known Issue)

Low Not Fixed

↳ 4.17 Incorrect Check Can Cause Array Index Out Of Bounds Error

Low Not Fixed

↳ 4.18 Different `uint` Type Used for Decimals in `Core` and `ERC20Plus` Smart Contracts

Low Not Fixed

↳ 4.19 The `max-burn` Function Returns an Incorrect Value

None Not Fixed

↳ 4.20 Unnecessary Waste of Gas

None Not Fixed

↳ 4.21 The `Value` Struct has Unused and Redundant Members

None Not Fixed

↳ 4.22 No Event Logged in the `mint` and `burn` Functions

None Fixed

↳ 4.23 Unnecessary Check in `is_legal_position` Function

None Not Fixed

↳ 4.24 Unnecessarily Lengthy Dynamic Array Declarations

None Not Fixed

5.0 Appendix A ——— 30

↳ 5.1 Severity Rating Definitions

6.0 Appendix B ——— 31

↳ 6.1 Thesis Defense Disclaimer

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.vy
- **Audit Commit:** af341ec9c6ee4a88d5bfd2437582b3f09a233930
- **Verification Commit:** b034c4b587cce7db2fa8d76f06da73b23726a516
- **Files in Scope:**
    - contracts/
        - core.vy
        - fees.vy
        - math.vy
        - oracle.vy
        - params.vy
        - pools.vy
        - positions.vy
        - token/
        - types.vy
- **Files Out of Scope:**
    - The `liquidate` function in `core` smart contract was considered out of scope.

Documentation

- None provided.

Section_2.0

Executive Summary

Schedule

This security audit was conducted from May 27, 2024 to June 12, 2024 by 2 security auditors for a total of 5 person-weeks.

Overview

This report details the findings and outcomes of our security audit of Velar Vyper Smart Contracts for Velar Perpdex. The Velar Vyper Smart Contracts are a series of smart contracts written in Vyper for the Bitlayer Bitcoin Layer 2. Velar Perpdex provide an external API for traders to trade for short and long positions using quote (stable coin) and base tokens. Liquidity providers can provide liquidity in base or quote asset to earn yield from fees collected from open positions in each block.

We performed an assessment of the overall security posture of the Velar Perpdex Vyper smart contracts, 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.

Threat Model

For the thread model of the Velar Vyper smart contracts, we adapted and updated the threat model that was used for the Velar Clarity smart contract implementation audit given the similarity in the implementations. We considered that the oracle that will be used by the Velar Perpdex will function as intended. We considered the following areas of concern and attack vectors in our audit:

- Front-running transaction causing traders to lose value;
- Closing a position not opened by the attacker, closing a position twice to get 2x rewards, or closing a position without penalty (in case of loss);
- Leveraging non-existent or incorrect tokens;
- Burning another user's tokens;
- Minting more value than deposited;
- Unlocking more reserves than intended by getting the base/quote-interest to 0;
- Leveraging incorrect fee calculations between short and long positions in malicious ways;
- leveraging static fee calculations to take control and deplete liquidity pools; and
- Making a malicious delegate call to bypass the `_INTERNAL` function check for smart contracts interacting with an external, unknown contract.

Security By Design

We found that protocol does not have any mechanism implemented to withdraw the protocol fees causing the protocol fees to get stuck in the core smart contract (Issue 7). Additionally, it is not guaranteed that liquidity providers will be able to withdraw their liquidity on demand. The protocol could delay withdrawing liquidity indefinitely (Issue 8).

The oracle mechanism and its implementation both heavily affect the security of the protocol. Through our discussions with the Velar team, we understand that the design of the oracle for the Velar protocol has not been finalized yet.

We have addressed two issues related to the Oracle smart contract where an issue in a slippage check results in traders' being unable to use open, mint, close, or burn operations (Issue 3). Additionally, a check for preventing signature replay results in only being able to perform one operation per block (Issue 6).

Governance

We found that the protocol does not implement a kill switch or a multistep process for transferring ownership of the protocol (Issue 12 and Issue 11).

Secure Implementation

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

We investigated potential issues in the handling of ERC20 token and found that the protocol does not update the total supply when burning tokens (Issue 1). Additionally, the protocol doesn't check the return value of token transfers which might cause incorrect token balancing (Issue 13). Moreover, the protocol doesn't handle fee-on-transfer tokens correctly (Issue 14). Also, the protocol doesn't handle ERC20Plus token decimals correctly (Issue 18).

Positions

We found some issues in handling positions in which a trader can repeatedly open and close positions within the same block, potentially creating an imbalance between the quote and base assets (Issue 2). Additionally, the protocol prevents closing valid positions by not considering trader's profit when deducting fees (Issue 9). We found an issue in incorrect calculation of positions fees that can result in depriving a position from receiving the related funding fees (Issue 10).

Precision Loss

We saw that there are various cases with precision loss in converting tokens and handling fees including We identified that a dishonest trader could create many low collateral positions across multiple addresses and bypass collateral fees (Issue 5).

Project Documentation & Tests

We found that the project does not have any documentation. This inhibited our audit as we did not have a reference of how the protocol was intended to work.

We found that there were tests for each of the contracts, as well as some scenarios. However, we did not find the tests to be comprehensive. For example, there is no test case for a scenario where price of an asset reaches to zero which can prevent closing of a position (Issue 4). Additionally, there is no proper testing of opening and closing a position in the same block which might result in an imbalance of quote and base reserves (Issue 2).

Use of Dependencies

We found there are no external dependencies or protocols included in the smart contract code. The protocol depends on a spot price which is received from the oracle. However, the oracle design and implementation has not been finalized yet at the current stage of the protocol. The protocol uses ERC20 tokens to represent quote and base tokens which are always considered to work as expected even though we found some issues relating to the use of those tokens in which there is no return value check when transferring tokens (Issue 13), and the protocol doesn't correctly handle fee-on-transfer tokens which can cause incorrect accounting (Issue 14).

Section_3.0

Key Findings Table

Issues	Severity	Status
ISSUE #1 Total Supply is Not Decreased While Burning Tokens	Critical	Fixed
ISSUE #2 Opening and Closing a Position in the Same Block Can Cause a Pool Imbalance	High	Fixed
ISSUE #3 Edge Cases While Checking Slippage Will Cause `price` Function to Revert (Known Issue)	High	Fixed
ISSUE #4 Long Positions Remain Open Even If the Asset Price Falls to Zero (Known Issue)	High	Not Fixed
ISSUE #5 Issues With Precision Loss	High	Not Fixed
ISSUE #6 Only One Operation Per Block Is Allowed Due to Oracle Price Check	High	Fixed
ISSUE #7 Protocol Fees Cannot Be Withdrawn (Known Issue)	High	Not Fixed
ISSUE #8 The Protocol Locks Reserves Indefinitely	High	Not Fixed
ISSUE #9 `calc_pnl_long` Function Can Prevent Closure of Valid Positions and Deprive the User From Receiving Profit	High	Partial
ISSUE #10 Incorrect Calculation of Positions Fees	High	Not Fixed
ISSUE #11 Protocol Owner is Not updated in Multi Step Operation (Known Issue)	Medium	Not Fixed
ISSUE #12 Protocol Does Not Have an Emergency Pause Mechanism (Known Issue)	Medium	Not Fixed
ISSUE #13 Unchecked Return Value From Token Transfer	Medium	Fixed
ISSUE #14 Protocol Does Not Handle Fee on Transfer Tokens	Medium	Not Fixed
ISSUE #15 Tokens Insufficiently Validated	Medium	Not Fixed

Issues	Severity	Status
ISSUE #16 Arbitrary Amount for Margin Protection (Known Issue)	Low	Not Fixed
ISSUE #17 Incorrect Check Can Cause Array Index Out Of Bounds Error	Low	Not Fixed
ISSUE #18 Different `uint` Type Used for Decimals in `Core` and `ERC20Plus` Smart Contracts	Low	Not Fixed
ISSUE #19 The `max-burn` Function Returns an Incorrect Value	None	Not Fixed
ISSUE #20 Unnecessary Waste of Gas	None	Not Fixed
ISSUE #21 The `Value` Struct has Unused and Redundant Members	None	Not Fixed
ISSUE #22 No Event Logged in the `mint` and `burn` Functions	None	Fixed
ISSUE #23 Unnecessary Check in `is_legal_position` Function	None	Not Fixed
ISSUE #24 Unnecessarily Lengthy Dynamic Array Declarations	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

Total Supply is Not Decreased While Burning Tokens

Critical Fixed

Location

tokens/ERC20Plus.vy#L102

Description

When a token is burned from a user's account using the `burn` function, it does not reduce the total supply of tokens. This disparity between the circulating total and the actual total number of tokens impacts accounting procedures for determining the quantity of LP tokens to mint and calculating the assets when removing liquidity.

Impact

Formula for minting tokens: `lp = (uv * ts) / pv` , where `uv` = user value

Formula for determining asset amount per LP token: `(lp * pv) / ts` ,

where `uv` = user value, `pv` = pool value, `ts` = total supply

To calculate the amount of tokens to be minted, the process involves multiplying `uv` by the total supply and then dividing by `pv` . With the total supply remaining constant, this leads to minting more tokens than originally expected. When liquidity providers remove liquidity from the pool, they receive fewer assets than anticipated due to the unchanged total supply.

Recommendation

We recommend reducing the total supply of lp tokens during the burning process.

ISSUE#2

Opening and Closing a Position in the Same Block Can Cause a Pool Imbalance

High Fixed

Location

positions.vy#L131

positions.vy#L307

Description

A trader can open a long position using the quote asset as collateral, upon closing this position, the trader will receive the base asset. Conversely, when a trader opens a short position using the base asset as collateral, they will receive the quote asset upon closing the position. Therefore, if a trader opens and closes a position within the same block, they will effectively swap the quote asset for the base asset in a long position and the base asset for the quote asset in a short position.

Impact

The `close` function lacks a check to prevent the closing of a position within the same block it was opened. This allows trader to repeatedly open and close positions within the same block, potentially creating an imbalance between the quote asset and the base asset. As a result, the pool might not have sufficient assets to pay traders when closing a position.

Recommendation

We recommend adding check which can prevent opening and closing the position in a same block.

```
def check(id:uint256) -> {

    pos: PositionState = Positions(self).lookup(id)

    assert block.number > pos.opened_at
   }
```

ISSUE#3

Edge Cases While Checking Slippage Will Cause `price` Function to Revert (Known Issue)

High Fixed

Location

oracle.vy#L86

Description

There are two edge cases in the `check_slippage` function that will cause the function to return false. This will cause the `assert 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 the current price is greater than the desired price, 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

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

ISSUE#4

Long Positions Remain Open Even If the Asset Price Falls to Zero (Known Issue)

High Not Fixed

Location

math.vy#L45

Description

When an asset price drops to 0, the `from_amount` function will revert due to division by zero, which prevents the user from closing the position.

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 revert caused by division by zero, these long positions will not be closed properly.

Recommendation

When closing the long position while price is zero, payout will be zero for user. We recommend not calling the `quote_to_base` function to convert the final amount which will prevent division by zero.

Verification Status

The Velar Perpdex team has not implemented the recommended fix stating that the behavior is intended.

ISSUE#5

Issues With Precision Loss

High | Not Fixed

Location

params.vy#L28

math.vy#L59-L64

math.vy#L51-L55

math.vy#L107

math.vy#L117

Description

There are numerous precision loss issues in the smart contracts including, but not limited to, the following cases:

1. If the user supplied collateral is less than 1000, the collateral fee will always be zero and for amounts greater than 1000, due to precision loss, the deducted fees will be lower than expected. The protocol has no way of knowing if different EOA addresses are controlled by the same trader. This means that an attacker 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 9,990, 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 precision loss, and the fees will be lower than expected.
2. 1. There is precision loss in converting quote tokens to base tokens and vice versa in `quote_to_base` and `base_to_quote` functions. Converting quote token USDT (6 decimals) to base token WBTC (8 decimals) with the current price of WBTC (~70000 USDT) has a three digit precision loss.
3. There are some precision loss issues in calculations in the `balanced` function in the `math` smart contract.

Impact

1. The user’s reduced fee payment will result in the protocol receiving less than anticipated.
2. The user’s virtual tokens are reduced by the amount of the precision loss at the time of closing a position, resulting in the user receiving less than expected. Liquidity providers are also affected the same way in that they receive less than expected at the time of withdrawal.

These precision losses can potentially lead to other unintended behavior, causing unexpected results.

Recommendation

We recommend reducing precision losses throughout the smart contracts especially in cases where there is more than one digit precision loss. This can be achieved by multiplying values to a specific constant value and dividing by the same value at the final stage of withdrawals.

Verification Status

The Velar Perpdex team stated that the behavior is intended, however they may decide to refactor the code in the future to accommodate more decimal points.

ISSUE#6

Only One Operation Per Block Is Allowed Due to Oracle Price Check

High Fixed

Location

oracle.vy#L62-L68

Description

The `verify_signature` function in the oracle smart contract prevents signature replay by recording the block number inside `SEEN` mapping and checking if the block has already been recorded in the mapping to disallow using the same signature twice in a specific block. However, this check results in allowing only one operation out of the mint, burn, open, or close in that specific block.

Impact

Only one of the four mint, burn, open, and close operations is allowed per recorded block.

Recommendation

We recommend using a different unique value instead of block number to avoid limiting operations performed in a single block.

ISSUE#7

Protocol Fees Cannot Be Withdrawn (Known Issue)

High Not Fixed

Location

contracts/oracle.vy

Description

The current implementation of the protocol doesn't include a `withdraw` function in the `core` smart contract to collect protocol fees.

Impact

Protocol fees will get stuck in the `core` smart contract.

Recommendation

We recommend implementing a withdraw mechanism to collect protocol fees.

ISSUE#8

The Protocol Locks Reserves Indefinitely

High Not Fixed

Location

pools.vy#L192

pools.vy#L196

Description

The protocol can prevent liquidity providers from withdrawing their liquidity from the pool on demand until an unspecified time if there is not enough unlocked reserves in the pool.

Impact

The protocol prevents liquidity providers from withdrawing liquidity when expected.

Recommendation

We recommend implementing a mechanism for liquidity providers to be able to withdraw their liquidity in a specific period in case there is not enough liquidity to fulfill instant withdrawals; for example by implementing a list of pending withdrawals using a queue or similar data structure and not allowing opening new positions with more than 1x leverage until all pending withdrawals are fulfilled by new reserves and closed positions. Additionally, we recommend clearly documenting the risk involved for liquidity providers so that they do not get caught by surprise at the time of withdrawing liquidity.

Verification Status

The Velar Perpdex team has not implemented the recommended fix stating that the behavior is intended.

ISSUE#9

`calc_pnl_long` Function Can Prevent Closure of Valid Positions and Deprive the User From Receiving Profit

High Partial

Location

positions.vy#L241-L242

positions.vy#L250

Description

Since a position's profit is calculated based on virtual tokens, the `calc_pnl_long` function in the `positions` smart contract prevents the closure of valid positions, as a result of arithmetic underflow, if the position has gained profit, however, the position collateral has been reduced to zero due to paying for borrowing and funding fees.

Impact

A valid position cannot be closed, and the user cannot withdraw profit.

Recommendation

We recommend taking user's profit into consideration when calculating the final PnL of the user to avoid preventing him from receiving his profit and being able to close the related valid position.

Verification Status

The Velar Perpdex team has partially implemented the recommended fix which only solves the problem of closing a position. However, this still can deprive the user from withdrawing their profit.

ISSUE#10

Incorrect Calculation of Positions Fees

High | Not Fixed

Location

positions.vy#L196-L221

positions.vy#L211

Description

The `calc_fees` function in the `positions` smart contract calculates the `funding_received` fees as zero if the `borrowing_paid` fees and `funding_paid` fees applied to the position collateral reduce the remaining collateral to zero. A trader might receive more fees than what has been paid. Therefore, the smart contract should first calculate the `funding_received` fees and then the `borrowing_paid` and `funding_paid` fees.

Impact

The smart contract deprives a position from receiving the related funding fees.

Recommendation

We recommend first calculating the received fees and then reducing the paid fees from the collateral-plus-fees-received.

Verification Status

The Velar Perpdex team has not implemented the recommended fix stating that the behavior is intended.

ISSUE#11

Protocol Owner is Not updated in Multi Step Operation (Known Issue)

Medium Not Fixed

Location

core.vy#L59

Description

The `set_owner` function sets the role 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:

```
 @external
def transfer_ownership(new_owner: address):
    self._check_owner()
    self.pending_owner = new_owner

@external
def accept_ownership():
    assert self.pending_owner == msg.sender, "Ownable2Step: caller is not the new
owner"
    self._transfer_ownership(msg.sender)
```

Verification Status

The Velar Perpdex team has not yet implemented the recommended fix. They stated their intention to implement the ownership in the `params` smart contract as a DAO.

ISSUE#12

Protocol Does Not Have an Emergency Pause Mechanism (Known Issue)

Medium | Not Fixed

Location

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.

```
 @external
def pause():
    self._check_owner()
    assert self._is_pause() == False
    self.Pause = True
```

Verification Status

The Velar Perpdex team has not implemented the recommended fix. They stated their intention to implement a mechanism which doesn't allow opening a new position in an emergency situation in the future.

ISSUE#13

Unchecked Return Value From Token Transfer

Medium Fixed

Location

core.vy#L200

core.vy#L243

Description

In the ERC20 interface, the `transfer` and `transferFrom` functions are implemented with return values to signify the outcome of their execution. These return values serve as indicators of whether the actions were successfully completed or not. Specifically, both functions return a boolean value: true denotes successful execution, while false indicates failure

The functions `open` , `close` , `mint` , and `burn` do not verify the return values from token transfers.

Impact

In the case of ERC20 tokens that return false instead of reverting when a transfer fails, this could lead to the protocol erroneously considering transfers as completed when they have not actually occurred.

Recommendation

Ensure to verify the return value of the token transfer.

ISSUE#14

Protocol Does Not Handle Fee on Transfer Tokens

Medium Not Fixed

Location

core.vy#L243

core.vy#L155

Description

When a transfer involves an ERC20 token implemented with the “fee on transfer” feature, the recipient will receive an amount that is less than the amount sent. The `core` smart contract may receive a token amount less than expected when a trader opens a position, or a liquidity provider provides liquidity.

Impact

The `core` smart contracts do not accurately check the amount received from trader and liquidity provider, this affects the overall value of the pool and the situation may arise where liquidity providers and traders are unable to withdraw their full assets from the pool.

Recommendation

In order to obtain the actual amount received by the contract, we recommend tracking the balance of tokens before and after the transfer of tokens:

```
 def transfer(amount) -> uint256:
   balanceBefore:uint256 = IERC20(token).balanceOf(core)
   IERC20Token(token).safetransferFrom(msg.sender, core, amount)
   balanceAfter:uint256 = IERC20(token).balanceOf(core)
assert balanceAfter >= balanceBefore
return balanceAfter - balanceBefore
```

Verification Status

The Velar Perpdex team has not implemented the recommended fix, however, they stated that they intend to implement it in the future.

ISSUE#15

Tokens Insufficiently Validated

Medium | Not Fixed

Location

core.vy#L112

core.vy#L157

Description

The `fresh` function in the `core` smart contract does not verify if it can mint the lp token at the `lp_token` address being passed to the function.

In case the `core` smart contract cannot mint the lp token, reserves cannot be added to the pool. Furthermore, another pool cannot be created with the same tokens since the `fresh` function prevents creating a pool with the same tokens twice putting the protocol in an undesirable state.

Impact

Reserves cannot be added to the created pool. A new pool with the same quote/base token pair cannot be created.

Recommendation

We recommend checking that the `core` smart contract can mint the token at the `lp_token` address passed to the `fresh` function before creating a new pool, by checking if the `core` smart contract is the owner of the `lp_token` .

ISSUE#16

Arbitrary Amount for Margin Protection (Known Issue)

Low | Not Fixed

Location

params.vy#L5

Description

The `is_legal_position` function checks whether the position 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:

```
 pool.base_reserves  >= (position.interest * 50) if position.long else (
pool.quote_reserves >= (position.interest * 50) )
```

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 have reserves to handle the position, but due to the above check, it fails.

Impact

The arbitrary multiplication of the position value by 50 times causes an incorrect assessments of the protocol's ability to handle a position. This may result in legitimate positions being rejected, even when the protocol has sufficient reserves. Meaning, users will be unable to open positions that are actually within the protocol's reserve limits. Resulting in an inefficient use of the pool's reserves.

Recommendation

We recommend refactoring the code to:

```
pool.base_reserves  >= (position.interest ) if position.long else (
  pool.quote_reserves >= (position.interest ) )
```

ISSUE#17

Incorrect Check Can Cause Array Index Out Of Bounds Error

Low Not Fixed

Location

positions.vy#L141

positions.vy#L87

Description

The referenced check in the `open` function in the `positions` smart contract can cause an array index out of bounds error if `get_nr_user_positions(user) == MAX_POSITIONS`.

Impact

The transaction reverts unexpectedly.

Recommendation

We recommend removing the equality check in the referenced `assert` statement.

ISSUE#18

Different `uint` Type Used for Decimals in `Core` and `ERC20Plus` Smart Contracts

Low | Not Fixed

Location

tokens/ERC20Plus.vy#L2

core.vy#L47

Description

The token decimal value is passed as `uint256` in the `ERC20Plus` smart contract and as `uint8` in the ERC20 interface within the core smart contract. If the decimal value in the `ERC20Plus` smart contract exceeds 255, attempting to fetch this value in `CONTEXT` function will cause a revert, since the `uint8` type can only handle values in the range of [0, 255].

Impact

The functions `open`, `close`, `mint`, and `burn` will not be able to call `CONTEXT` because attempting to fetch the decimal value of the quote and base assets in `CONTEXT` will cause a revert:

```
 @internal
def CONTEXT(base_token: address, quote_token: address, op: OraclePrice) -> Ctx:
  return Ctx({
    price         : self.ORACLE.price(op),
    base_decimals : convert(ERC20Plus(base_token).decimals(), uint256),
    quote_decimals: convert(ERC20Plus(quote_token).decimals(), uint256),
  })
```

Recommendation

We recommend using same `uint` type for decimal value in the `ERC20Plus` smart contract and in the ERC20 interface present in the `core` smart contract.

ISSUE#19

The `max-burn` Function Returns an Incorrect Value

None Not Fixed

Location

pools.vy#L208

Description

The `max-burn` function calculates the maximum amount of lp tokens to burn using the formula `(uv * total_supply) / (pv - 1)`, where `pv` is the pool value (total reserve) and `uv` is the unlocked value. However, decreasing `pv` by 1 during this calculation results in an incorrect amount of lp tokens.

Impact

None – no security impact.

Recommendation

We recommend calculating the amount of lp token using `(uv * total_supply) / (pv)`

ISSUE#20

Unnecessary Waste of Gas

None Not Fixed

Location

fees.vy#L128-L140

Description

In the `update` function in the fees smart contract, when `new_terms` is zero, the referenced lines of code are executed unnecessarily, wasting a substantial amount of gas.

Impact

None – no security impact.

Recommendation

We recommend checking the `new_terms` value and only execute the aforementioned lines of code if it is greater than zero.

ISSUE#21

The `Value` Struct has Unused and Redundant Members

None Not Fixed

Location

types.vy#L19-L30

Description

`base` , `quote` , `base_as_quote` , `quote_as_base` , `total_as_base` , `quote_excess_as_base` members of the `Value` struct are unused, and therefore redundant and should be removed.

Impact

None – no security impact.

Recommendation

We recommend removing the aforementioned struct members to save gas and improve readability.

ISSUE#22

No Event Logged in the `mint` and `burn` Functions

None Fixed

Location

tokens/ERC20Plus.vy#L95

tokens/ERC20Plus.vy#L102

Description

The `mint` and `burn` functions update the balance of an address, but they do not log any corresponding events within these functions.

Impact

This can affect various off-chain tools, monitoring systems, and services that depend on events to accurately track real-time contract activities. Detailed and informative events are essential for security monitoring, allowing projects to respond effectively.

Recommendation

We recommend logging events for the `mint` and `burn` functions.

ISSUE#23

Unnecessary Check in `is_legal_position` Function

None | Not Fixed

Location

params.vy#L7

Description

The function `is_legal_position` verifies if the leverage is 10 or less by iterating through values from 1 to 10. When the leverage value is 1, the execution cost is 167 gas, and as the leverage value increases, the execution cost also rises.

```
 @external
@pure
def is_legal_position(leverage: uint256) -> bool:

  lev : bool = leverage == 1 or (
               leverage == 2 or (
               leverage == 3 or (
               leverage == 4 or (
               leverage == 5 or (
               leverage == 6 or (
               leverage == 7 or (
               leverage == 8 or (
               leverage == 9 or (
               leverage == 10 )))))))))
  return  lev
```

Impact

It costs more gas to iterate through every value from 1-10.

Recommendation

We recommend using the code below to check leverage, which is more gas-efficient. Regardless of the leverage value, the execution cost will remain constant at 139 gas:

```
 @external
@pure
def is_legal_position_test( leverage: uint256) -> bool:

  lev : bool = leverage <= 10
  return lev
```

ISSUE#24

Unnecessarily Lengthy Dynamic Array Declarations

None | Not Fixed

Location

types.vy#L129-L137

Description

Members of the referenced struct are declared as dynamic arrays with a length of 100. However, the maximum number of indexes used for those dynamic arrays is 2. This brings redundancy and results in unnecessary waste of gas and degrades readability. Reading from a 100 length dynamic array substantially increases gas usage in comparison to reading from a 2 length dynamic array in a transaction.

```
struct Deltas:
  base_interest   : DynArray[Instr, 100]
  quote_interest  : DynArray[Instr, 100]
  base_transfer   : DynArray[Instr, 100]
  base_reserves   : DynArray[Instr, 100]
  base_collateral : DynArray[Instr, 100]
  quote_transfer  : DynArray[Instr, 100]
  quote_reserves  : DynArray[Instr, 100]
  quote_collateral: DynArray[Instr, 100]
```

Impact

None – no security impact.

Recommendation

We recommend declaring the length of the dynamic arrays explicitly to the numbers indexes used.

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 • Permanent freezing of NFTs • Unauthorized minting of NFTs • Predictable or manipulable RNG that results in abuse of the principal or NFT • Unintended alteration of what the NFT represents (e.g. token URI, payload, artistic content) • 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 does not 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.