hack·a·chain

SECURITY AUDIT REPORT

in favor of CARMINE FINANCE

Summary

This report has been prepared for Carmine Finance in the source code of the project as well as in project dependencies that are not part of an officially recognized library. The audit has been conducted by combining static and dynamic code analysis with a manual review of the source code by Hack-a-Chain's research team.

The audit process analyses:

1) Adherence to widely recognized best practices and industry standards;
2) Vulnerability to most common attack vectors;
3) Thorough line-by-line review of the code base;
4) Ensuring that contract logic meets the specifications of the project's whitepaper.

The security audit result is composed of different findings, whose vulnerabilities are classified from critical to informational, according to the following impact versus likelihood matrix:

Impact	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Informational

Likelihood

After presenting the findings to the client, they are granted a 7 days period to fix the vulnerabilities. This report will specify all vulnerabilities found and whether they were fixed by the team.

Overview

Project Summary

Project Name	Carmine Finance
Auditor	Joao Veiga, Pedro Destri
Description	Options Automated Market Maker
Platform	Starknet
Language	Cairo
Codebase	https://github.com/CarmineOptions/carmine-protocol
Initial Commit	6274ab7b2b4a0bfd5d524b67f9996071bab8e319
Commit after fixes	4920a68930816a43b0d0aeaf85a1688a58c4aba0

Audit Summary

First delivery date	04/05/2023
Final delivery date	05/01/2023

Code Inventory

AMM	AMM contract

Findings

ID	Title	Category	Severity	Status
AMM-1	Handle stale oracle prices	External contract	Low	Fixed
AMM-2	Diverging pool-oracle assets	Logical	Medium	Fixed
AMM-3	Reentrancy guard consistency	Reentrancy	Informational	Fixed
AMM-4	Pricing model documentation	Documentation	Informational	Acknowledged
AMM-5	Test suite documentation	Documentation	Informational	Acknowledged

Table of contents

Audit report

The audited code consists of 4 separate smart contracts: (1) options AMM, (2) Proxy implementation, (3) token representing option contract, (4) token representing liquidity pool shares, available at https://github.com/CarmineOptions/carmine-protocol.

Sections 1, 2, and 3 perform a thorough overview of the contracts logic, architecture, and implementation.

Section 4 provides a specific description of each audit recommendation.

The disclaimers section provides the legal terms of responsibility for the auditor.

1 Proxy

The application utilizes the Proxy pattern, which allows upgrades in the contract's code to be made seamlessly. The contract is split into two parts (1) a Proxy contract that stores the application's state and redirects all calls it receives to (2) an Implementation contract, which defines the logic for the contract.

The core aspect of the Proxy pattern is that it allows the implementation contract to create a function to upgrade the contract - which is done by changing the address of the implementation contract that is saved in the Proxy contract's state.

1.1 Implementation

The Proxy implementation (contracts/proxy_contract/proxy.cairo) follows Openzeppelin's standard for Cairo Proxies, mimicking a widely tested external library.

However, the utilization of the Proxy pattern also requires a different implementation in the logic contract than a regular contract would:

1. It cannot set its initial state through a regular constructor, it must have a different external function that calls Proxy.initializer
2. It must have a function to upgrade the reference to the implementation contract (if the contract is to be upgradable)
3. The contract must protect the upgrade function so that only an authorized account can call it

After reviewing the code implementation, we found that all those requirements were fulfilled.

Another important point of attention is storage collision. In traditional Proxy architectures on EVM, upgrades of the contract can clash storage locations based on the order of declaration of variables (this is a system known as structured storage). Starknet implements **unstructured storage** by default, meaning the memory location of every storage variable is a hash of its name. This makes it extremely difficult for 2 variables to collide in the case of an upgrade.

The team has also been extra careful and implemented a series of tests (tests/test_proxy.cairo) to assert that storage collisions do not happen.

1.2 Dependencies

The Proxy contract is highly dependent on the OpenZeppelin library for Cairo. The library claims to be *highly experimental code* in its repository and advises users to *use it at their own risk*.

Even though that is a point of concern, we have reviewed the Proxy contract implementation from OpenZeppelin and extensively read its documentation. The logic is robust and the studies made by the team have shaped some of the most important design patterns found for Cairo contracts, which is why we deem its use to be secure.

2 Token contracts

The protocol implements 2 different token contracts following the ERC-20 token implementation from the Openzeppelin library for Cairo contracts.

- lptoken (contracts/erc20_tokens/lptoken.cairo) -> Represents shares of ownership of the liquidity pools in Carmine
- option_token (contracts/erc20_tokens/option_token.cairo) -> Represents an option contract owned by a user

lptoken is implemented as a regular ERC20 token with no extra capabilities.

option_token is implemented a little differently. It stores the option's metadata in its state: base token, quote token, option type, strike price, maturity, and side.

2.1 Dependencies

The contracts are implemented following the design pattern of Openzeppelin's library for token contracts. The library claims to be *highly experimental code* in its repository and advises users to *use it at their own risk*.

Even though that is a point of concern, we have reviewed the token implementations from Openzeppelin and extensively read its documentation. The logic is robust and the studies made by the team have shaped some of the most important design patterns found for Cairo contracts, which is why we deem its use to be secure.

Moreover, the importance of token contracts for the overall security of the contract is minimal, since they are only responsible for representing token logic and do not implement any liquidity pool or option logic - those are entirely structured inside the Automated Market Maker Contract.

3 Automated Market Maker

The Automated Market Maker contract (AMM) is responsible for implementing the core logic for Carmine. The construct is structured as a logic contract, to be used with a Proxy contract (refer to Proxy section).

3.1 A review of option contracts

The contract allows traders to buy or sell options contracts.

An option is a derivative contract between two parties that grants its holder (the trader who bought the option) the right - but NOT the obligation - to buy or sell its underlying asset at a fixed price (strike price) on a specific date (maturity)[1].

Calls are the name given to options that give their holder the right to purchase the underlying asset at a given price. For traders to make money with a call, the price of the underlying asset must rise above the option's strike price - which would allow the holder to buy the asset at a discount and immediately sell it on the open market.

Puts are the name given to options that give their holder the right to sell the underlying asset at a given price. For traders to make money with a put, the price of the underlying asset must fall below the option's strike price - which would allow the holder to buy the asset at market price on the open market and immediately exercise the option's right to sell it at the strike price.

In practice, most options markets implement cash-settlement. Cash-settlement is a type of settlement that involves paying or receiving cash instead of actually buying or selling the underlying asset - this is the settlement used in Carmine Finance.

3.2 Carmine implementation of options

Options are traditionally traded in derivative exchanges through order books or Over the Counter in traditional financial markets. Blockchains are notorious for being order book unfriendly and most trading markets in DeFi are run by Automated Market Makers.

Automated Market Makers are algorithms that are always ready to provide a sale price for any trader wanting to buy or sell an asset - they will always accept the trade and assign a given price to it based on a mathematical formula. This is the

[1] This is actually the description of an european style option. American style options allow its holder to exercise the buy/sell right at any time before expiration.

design that powers Uniswap, Balancer, and a large quantity of DeFi powerhouses that operate spot trading markets.

However, there is a huge difference between derivatives markets (such as options) and spot markets. In derivatives markets it is not possible to simply trade assets, each asset represents the obligation of a party to pay an undetermined amount of money to the other party based on uncertain future events. Therefore, Carmine must implement a way to keep option sellers liable for the options that they have sold.

The solution found by Carmine is to have 2 different liquidity pools for each pair of assets - one for calls and the other for puts.

1. When a user wants to buy an option, the liquidity pool is the seller (counterparty)
    a. liquidity pool receives premium
    b. liquidity pool locks capital to guarantee possible option settlement
    c. user receives option token representing the call
    d. at settlement, if the option is in the money, the user receives their profits and the pool unlocks the rest of the locked capital
2. When a user wants to sell an option, the pool buys it from them
    a. user receives premium
    b. user must lock capital in the AMM contract to guarantee possible option settlement
    c. user receives option token representing their position
    d. at settlement if the option is in the money, pool receives their profits and user unlocks the remaining capital

Using this mechanism the pool is always able to provide trades to any user - given that liquidity is sufficient to lock capital at risk/pay premium.

The key component in this mechanism is the pricing formula used to determine the price of options at any given time, which is analyzed in the next section.

3.3 Pricing formula

Spot market AMMs traditionally utilize very simple pricing formulas, such as the constant product formula. Options pricing, on the other hand, is very complex mathematically. Carmine chose to utilize the Black-Scholes model, one of the first and most widely used option pricing models as the formula that controls the AMMs dealings.

Call option

$$C(S_t, t) = N(d_1)S_t - N(d_2)Ke^{-r(T-t)}$$

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}[ln(\frac{S_t}{K}) + (r + \frac{\sigma^2}{2})(T-t)]$$

$$d_2 = d_1 - \sigma\sqrt{T-t}$$

Put option

$$P(S_t, t) = Ke^{-r(T-t)} - S_t + C(S_t, t)$$

$$= N(-d_2)Ke^{-r(T-t)} - N(-d_1)S_t$$

Where:

- C(St, t) - Price of Call at time t and underlying price St
- P(St, t) - Price of Put at time t and underlying price St
- N - Standard normal cumulative distribution function
- S_t - Price of the underlying asset at time t
- K - Strike price
- r - Annualized risk-free rate
- T - Time of option expiration
- t - Current time
- σ - Volatility of S_t

Analyzing the relevant variables we discover that most values can be precisely determined at any point in time by searching external data sources:

- S_t can be queried from spot markets
- K is fixed for each option type
- r can be queried from a selected central bank rate (Carmine actually sets it as a constant for each option type upon creation)
- T is fixed for each option type

- t can be queried from the current block's time

Indeed all those values are fixed, except for S_t which Carmine queries from an on-chain oracle (see oracle section for security concerns).

The calculation of σ is the most innovative part of the protocol. Even in traditional finance, there is a high degree of disagreement on how to calculate σ and finance practice discourages the usage of implied volatility from the Black-Schole model given its assumption of lognormal price distributions.

Instead of using the volatility of S_t, Carmine utilizes the volatility of the option itself inside the protocol as its proxy. This allows the price to respond to trading volume even if the underlying asset's oracle price remains constant.

This design requires volatility for Puts and Calls and options of different maturities to be calculated separately.

The formula used to calculate volatility is:

$$\sigma = \frac{\sigma_{t-1} + \sigma_t}{2}$$

$$\sigma_t = f(\sigma_{t-1}, PS_t, Q_t) = \sigma_{t-1} + \frac{Q_t}{C}$$

Volatility is defined as the average between previous volatility and current volatility.

The current volatility is calculated based on:

- C - constant determining the speed at which the volatility changes
- Q_t - size of trade at time t - positive for longs and negative for shorts)

It is important to distinguish between using Black-Scholes to price an option and using Black-Scholes to update an option's price. The key difference is the way in which volatility is estimated.

Considering Black-Scoles' theoretical background, whenever the model deals with volatility it is referring to the expected future volatility of the underlying asset. This is a theoretical value that has to be predicted by whoever is using the model - common proxies used in financial markets are past volatility and volatility indexes. It is also important to note that, whenever we look at an option's current market price it would be theoretically possible to derive its implied volatility by applying the model - however, such derivation does not work in practice since Black-Scholes assumes a lognormal distribution, whilst market agents tend to price expected volatility with skewness and kurtosis in the distribution curve.

Carmine's pricing algorithm already takes these concepts into account and uses Black-Scholes as a price updating mechanism. By updating the volatility with every trade performed, the algorithm is able to adjust supply and demand even if all oracle feeds and model variables remain constant. It is necessary, however, to be very careful when tuning the model's parameters for new options as price divergences from price-discovery markets[2] generate arbitrage opportunities. There is no problem *per se* with arbitrage on the protocol as it is necessary to maintain price parity, however, the algorithm must be optimized to converge to correct prices as easily as possible since every arbitrage trade represents a loss for liquidity providers.

Through mathematical modeling, we were unable to find vulnerabilities in the protocol. However, we recommend caution in the deployment of the application as it is not possible to predict how market agents and arbitrage bots are going to react to the AMM. The team is taking the correct approach by phasing the protocol's deployment and establishing trade limits to ensure the safety of liquidity providers' funds.

3.4 External contract dependencies

Interoperability between smart contracts is a major source of risk. Bugs, exploits or errors in dependency smart contracts can cause unpredicted problems for the protocol.

Carmine only integrates one external contract - the Empiric Oracle solution to provide the prices of underlying assets.

Carmine fetches price data from the oracle using 2 different functions:

[2] Price-discovery markets are highly liquid markets where most of the trade of a certain instrument takes place. Usually prices are formed in this market and other less liquidy markets follow through arbitrageurs trades.

- empiric_median_price -> calls get_spot_meadian on oracle - which returns the current median price from all spot sources
    - This is used to price the option before expiry
- get_terminal_price -> calls get_last_spot_checkpoint_before on oracle - which returns last checkpoint value before option expiry
    - This is used to price the option in maturity

Empiric is designed to be a decentralized oracle, with multiple data sources for maximum availability and trustworthiness and has been audited by multiple independent auditing firms. All these reasons validate the trust that the project has in the oracle.

However, there are two situation in which the oracles functionality might be compromised:

Network congestion

In the event of network congestion, it is possible that data publishers of Empiric are unable to post their data in time, which is going to cause the oracle's spot price to be outdated.

In the scenario of persistent outdated prices, traders are able to take a large number of trades until the volatility parameters compensate for the price. This is going to cause a huge loss for the liquidity pool and might lock a large percentage of the pool's capital, further hindering withdrawals.

Empiric Oracle has a mechanism to help protocols deal with this problem, every query to the Oracle returns last_updated_timestamp. Validating that price feeds are not stale for a long time is a good way to prevent huge losses for the pool, in case of stale prices trading could be halted until oracles update prices again.

Asset depegging

The Oracle price feed uses USD, while the protocol's pools are denominated in USDC. This is not a problem for pools based on base assets (Call pools) since they essentially represent a real ETH/USD option with no influence from any stablecoin.

On the other hand, Put pools are denominated in the quote asset, which might cause confusion. In the case of a stablecoin deppeging from its target price, the option's prices and payouts are going to be distorted.

This situation introduces a lot of financial complexity and can cause unintended consequences including:

1. Unexpected behavior of portfolio when on-chain traders use options as a hedge for their positions
2. Possibilities of huge flash loan predatory arbitrages in depegging events

3. Mathematical complexity in modeling option payouts and fair value given stablecoin risk

In our view, a simpler approach would be preferred, by tracking the price of ETH/USDC from oracles (this is possible by composing the ETH/USD and USDC/USD oracles offered by Empiric).

3.5 Team indicated concerns

An important aspect of security reviews is to understand the domains of knowledge that the team is most comfortable with and which parts of the code are more heavily battle tested and which have undergone a lot of iteration and recent updates.

After asking the project's team about the areas they were more eager to see tested through the audit they have pointed out the following:

1. Eventual Starknet Renegenesis issues
2. Recent updates on volatility calculation
3. Recent updates on reentrancy protection
4. Usage of Black-Scholes as price updating mechanism (not price discovery mechanism)
5. Impossibility to compute the pool's valuation in specific cases causing the trading of the option to be locked

The team also mentioned a series of usage issues related to front-end and testnet deployment which were kept out of the report as they were irrelevant to the security audit.

We have evaluated each of the individual requests:

Eventual Starknet Regenesis issues

Starknet still runs as an Alpha version. At this point, the blockchain's Virtual Machine (VM) runs both Cairo 0.x versions[3] and Cairo 1.0. However, the planned roadmap is to entirely eliminate support for 0.x versions of Cairo in the future, to simplify the development and usage of the network.

This event is called Regenesis and once it happens all existing Cairo 0.x contracts in Starknet are going to stop working.

[3] 0.x refers to all minor versions before Cairo 1.0

The network is currently in a Transition period, in which Cairo 1.0 is being rolled out. Cairo 1.0 has not yet reached feature parity with Cairo 0.x, but once it does, it is going to be possible/necessary to upgrade all existing contracts to Cairo 1.0.

As explained by Starkware's team in this thread, a further update to Starknet is going to introduce a special syscall to upgrade existing contract addresses' code to a Cairo 1.0 version. All existing applications are required to perform such a migration lest they are going to stop working as Regenesis goes live.

After carefully reviewing the planned Regenesis architecture and requirements on official documentation and Starknet's forums we have come to the conclusion that there is no risk for Carmine as long as Regenesis happens in the way that it is planned.

Since Carmine already implements a Proxy-based architecture, it must execute the following steps for the contract to work after Regenesis:

1. As soon as the Proxy contract is available in Cairo 1.0, the team must alter the implementation contract to include a migrate function that would migrate the Proxy's code to a new class_hash corresponding to the Cairo 1.0 implementation;
2. The team must then point the proxy to the new implementation and call migrate function;
3. Now Proxy is already using Cairo 1.0, although implementation is still using 0.x. The team must now rewrite the implementation code using Cairo 1.0, deploy it and point Proxy to its new class_hash. This step can be performed even after the Regenesis (but the contract is going to be stuck from Regenesis until this upgrade is made)

The only precaution needed from the team is to keep up with Starknet's announcements regarding Regenesis to guarantee that every necessary measure is taken at the appropriate dates.

Recent updates on volatility calculation

Volatility values were previously considered the same for options trading in the same liquidity pool and with the same maturity date. This meant that options with different strike prices would always consider the same volatility parameters. This setup has been changed and now each different option has its volatility calculated separately.

If we consider the efficient market hypothesis[4], options with the same maturity and underlying asset should have the same implied volatility. Nevertheless, in real markets, different implied volatilities are observed for different strike prices. The new volatility model is better as it allows the market to determine the implied volatility of the underlying asset instead of forcing the theoretical equivalence.

We have not found any conceptual problem when looking at such a scenario. These arbitrage opportunities are moderate and in line with the expected arbitrage to bring on-chain prices in line with those practiced in price-discovery markets.

Upon analyzing the code implementation of the new volatility calculation we have run fuzzing tests on the contract to guarantee results are correct for every different pool. We were not able to find any inconsistencies with the intended behavior.

Recent updates on reentrancy protection

The team has introduced the reentrancy guard pattern from Openzeppelin's library into the AMM contract in 3 different functions:

- do_trade
- close_position
- expire_option_token

The pattern is relevant here since calling these 3 functions (which are actually internal functions called from other @external functions) results in the transfer of ERC20 tokens between accounts (calls to external contracts not controlled by the project).

After analyzing the functions, we have identified that the contract only ever calls 4 different contract interfaces:

1. Empiric oracle
2. LP tokens
3. Option tokens
4. ERC20 tokens

The main risk of reentrancy attacks happens when the contract calls an untrusted contract that might hijack the application flow and implement malicious logic.

In this case, only Empiric oracle and ERC20 tokens are external contracts.

Empiric oracle is a fixed contract address that is already trusted by Carmine for very important processes. In this case, reentrancy is not a risk.

[4] The efficient market hypothesis considers a market in equilibrium, where there are no arbitrage opportunities.

ERC20 contracts are where the biggest risk lies. The addresses accepted by the contract are imputed when new pools are initialized. In case a pool is initialized with a token contract that is malicious, there is a risk of reentrancy being used. Nevertheless, this risk is very low as the protocol needs to vet every token contract that is accepted.

Overall it is a good practice to implement reentrancy locks when calls to undetermined contracts are made, but there are instances where calls to ERC20 contracts are made without implementing reentrancy locks: withdraw_liquidity, deposit_liquidity. We suggest implementing the locks in these functions to follow the pattern for every untrusted external contract call.

Usage of Black-Scholes as price updating mechanism

There can be a series of problems when using Black-Scholes as a pricing mechanism for options. In traditional markets, the Black-Scholes model does not always represent the observed market prices for a variety of reasons. If the AMM were to simply use Black-Scholes for pricing it could end up having its entire liquidity drained by arbitrageurs when the model's prices diverge from prices found in more liquid markets.

Carmine uses Black-Scholes instead as a price-updating mechanism. The key factor here is the usage of the option's volatility as the volatility parameter in the model. The Black-Scholes model together with the volatility updating formula describes the pricing curve for the AMM - similar to how AMMs for spot markets function - supply and demand are balanced through the volatility update.

Any flaw of the Black-Scholes model that can be considered relevant when pricing options - such as distribution assumptions - is not relevant here since the AMM is going to balance supply and demand by updating the volatility and Black-Scholes merely gives out the shape of the curve.

Impossibility to compute pool's valuation in specific cases causing the trading of the option to be locked

The current implementation of Black-Scholes and Cumulative Distribution Function of the normal distribution currently has a problem when dealing with too distant strike prices (option too much in or out of the money) or too close to expiration. Under those conditions, the model is not able to converge to an answer within the blockchain's limitations, causing trading and liquidity deposits/withdrawals to halt.

This is a recognized limitation of the system that the team intends to improve on further versions of the app. It is relevant nevertheless for the scope of the audit to understand if a malicious actor can abuse this limitation of the code to cause financial damages to the protocol.

The hypothesis explored during the audit were:

1. Exploiters forcibly pushing the price of the option to lock the pool's capital
2. Fuzzing parameters to find odd relations that would also break the pricing mechanism

Hypothesis 1 was focused on more broad DeFi attacks. Let us suppose a liquidity provider in Carmine also holds a large undercollateralized position in a lending protocol. Attackers could buy a huge chunk of options in the same block to lock all liquidity in Carmine and stop the provider from recovering capital to recollateralize the position, thus forcing a liquidation.

After exploring this scenario we found it very unlikely given that (1) liquidity providers are aware of the possible locking of their capital and (2) this attack would carry a large economic cost to move the price of the options, especially if liquidity in the pool is abundant. We thus did not consider it viable

Hypothesis 2 was tested through fuzzing on the Black-Scholes implementation and comparing it against results on native non-blockchain based implementations. Our tests did not find significant results showing possibilities of having trades/liquidity locked outside of the cases already expected.

Overall, although a problem that must be improved upon in the next iterations of the protocol, the problem in computing options prices did not create any viable exploit options.

3.6 Generic Vulnerability Hypothesis

Besides the specific studies highlighted above, checking for common vulnerabilities found in blockchain and traditional attack vector repositories is also a very relevant part of the audit process.

For this project we have focused on testing for the following vulnerabilities:

1. Overflows, Underflow, and arithmetics bugs
2. Access control implementation
3. Race conditions

Overflows, Underflow, and arithmetics bugs

Arithmetic bugs are a big concern for smart contracts that deal with complex mathematical operations as is the case for Carmine Finance.

Starknet also magnifies the problem by adding the complexity of finite fields.

Carmine adds even more complexity through its use of Math64x61 library for fixed-point arithmetics.

Considering the stacked layers of complexity found in the contract's mathematical operations, the team opted for an integration-based approach to catch bugs. This is achieved by performing input fuzzing on the main contract functions and checking that all results are the intended ones by comparing them against expected results.

We were unable to find any inconsistencies besides those already documented for the Black-Scholes model calculation.

Access control implementation

Failed implementations of access control can lead to disastrous consequences.

The contract implements access control through the Proxy contract, which is secure. In the setup, it only allows alterations of ownership if they are signed by the owner and all protected functions assert their origin from the owner.

Race conditions

Race condition bugs arise when the ordering of transactions sent to the contract alters their outcome in unintended ways.

The main situation in which this affects DeFi protocols is by allowing bots to front-run legitimate traders and profit from their price movements. This phenomenon cannot be entirely prevented by the protocol design itself and is also present in centralized markets.

The contract implements parameters to control slippage and deny transactions in case their final cost surpasses a user-defined parameter. We find this to be in line with best practices and sufficient to prevent considerable user losses due to front running.

4 Detailed issue report

AMM-1 - Handle stale oracle prices

Category	Severity	Location	Status
External contract	low	oracles.cairo	Fixed

Description

The Empiric oracle works by aggregating price information from multiple trusted sources that push data on-chain in constant time intervals.
When get_spot_median is called on the oracle it returns the current price, the timestamp when the price was last updated and the total number of data sources aggregated for the call.

Even though the Empiric oracle network is designed for maximum availability and trustworthiness, it is possible that network congestion, corruption of agents, and other factors harm the speed of data aggregation. This causes the price in the oracle to become outdated (stale).
In cases where the price becomes stale a lot of profitable arbitrage opportunities will arise in the AMM, prompting traders to take advantage of them. This spikes volatility in the AMM's formula. Once the price feeds go back online, new huge arbitrage opportunities will arise, as the price in the formula abruptly changes and volatility remains spiked for the first trades.
Large enough times of stale prices or oracle intermittency might generate a lot of huge arbitrages that end up hurting liquidity providers.

Recommendation

To avoid unintended consequences of having trades settled using stale prices over long time periods it is recommended to implement a transaction lock whenever oracle prices are found to be stale. The exact time limit for prices to be considered stale should be defined for each feed, according to the feed's characteristics.

Alleviation

Team implemented a lock on trading if oracle prices go stale for over an hour. This measure greatly reduces the risk of abusing congested network situations and manages to maintain high availability of the protocol.

AMM-2 - Diverging pool-oracle assets

Category	Severity	Location	Status
Logical	Medium	oracle.cairo	Fixed

Description

The current logic of the contract queries the oracle for prices using USD as the quote asset (e.g. ETH/USD), however, the liquidity pools for trades are implemented using USDC as the quote asset (e.g. ETH/USDC).
Even though USDC is a stablecoin that aims to be pegged 1:1 to USD it is possible that market volatility, insolvency issues, and high demand for stables/fiat alter its value relative to USD. This kind of issued has been seen repeatedly in the last 12 months and cannot be ignored by DeFi protocols anymore.
This setup does not affect call pools as they are denominated in the base asset only. On the other hand, put pools are going to suffer from inconsistency. In depegging scenarios, DeFi hedging positions might be rendered useless as the pool uses USD for quoting and USDC for payments. Moreover, in the complex fallouts of a congested network over black-swan kinds of events, the financial implications of the confusion between assets might be the driver for liquidations and other unpredicted financial consequences.
Overall the current setup can have unintended consequences on depegging events, especially when coupled with network congestion and panic sales that usually accompany such events.

Recommendation

We advise the use of USDC as the quote asset when fetching prices from the oracle. This can be obtained by fetching both ETH/USD (or whichever base token is being used) and USDC/USD quotes from the oracle and performing the necessary calculations.

Alleviation

Team implemented the recommended changes, patching the vulnerability.

AMM-3 - Reentrancy guard consistency

Category	Severity	Location	Status
Reentracy	Informational	liquidity_pool.cairo	Fixed

Description

The code is currently applying the ReentrancyGuard pattern from the Openzeppelin library on the following functions:

- do_trade
- close_position
- expire_option_token

These are all functions that perform calls to external unverified contracts (ERC20 tokens), which could theoretically produce a reentrancy problem.

However, there are 2 other functions in the contract that also perform similar operations and do not implement the reentrancy protection:

- deposit_liquidity
- withdraw_liquidity

Recommendation

We advise the inclusion of reentrancy protection in all functions that perform external contract calls to unverified contracts to maintain the security best practice and consistency across the project.

Alleviation

Team implemented the reentrancy guard at every instance where there was a call to an external undefined contract.

AMM-4 - Pricing model documentation

Category	Severity	Location	Status
Documentation	Informational	Documentation	Acknowledged

Description

The project carries a lot of mathematical complexity in its implementation. An unaware reader of the documentation without access to the project team's expertise is likely to not be able to comprehend the rationale behind the pricing model used in the protocol.

Moreover, explaining the technical differences between using the model as a price-adjusting mechanism through the option's volatility and using it as a general price model might induce the reader to believe that the protocol uses incorrect financial concepts.

Recommendation

We advise the inclusion of special documentation regarding the intended behavior of the Black-Scholes and volatility models used in the contract. In particular, the updating aspect of the model and its difference from the traditional use of Black-Scholes in order-book-based trading should be explained.
It is also advisable to explain the rationale behind selecting such an algorithm and its expected behavior in different stress scenarios.

Adding overall contract documentation explaining the technical decisions and architecture would also be a good practice to facilitate new developers and auditors when first getting acquainted with the code.

Alleviation

The team acknowledged the issue, however chose not to implement the improvements at this point as they do not pose any security threat. Team is going to improve documentation over time as their technical resources become available for non essential work.

AMM-5 - Test suite documentation

Category	Severity	Location	Status
Documentation	Informational	Documentation	Acknowledged

Description

The project has an extremely high-quality test suite in place performing a series of important assertions to guarantee the correct functioning of the contracts.

However, to external auditors, developers and onlookers the test suite is extremely hard to read and is not always clear about its intentions. Goos testing documentation states (1) exactly what is - and what is not - being tested within each test, (2) the organization of the test folder, (3) how and where to add new tests when necessary.

Recommendation

We advise the inclusion of general documentation for the testing suite, explaining the overall architecture and how to add new tests to it. Inside each file, documentation should be improved to be more clear about exactly what is being tested, why, and how.

Alleviation

The team acknowledged the issue, however chose not to implement the improvements at this point as they do not pose any security threat. Team is going to improve documentation over time as their technical resources become available for non essential work.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer, and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Hack-a-Chain's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Hack-a-Chain to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intended to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Hack-a-Chain's position is that each company and individual are responsible for their own due diligence and continuous security.

Hack-a-Chain's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Hack-a-Chain are subject to dependencies and are under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are

emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCTS, OR OTHER MATERIALS,OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED “AS IS” AND “AS AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, HACK-A-CHAIN HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, HACK-A-CHAIN SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, HACK-A-CHAIN MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER’S OR ANY OTHER PERSON’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, HACK-A-CHAIN PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED. WITHOUT LIMITING THE FOREGOING, NEITHER HACK-A-CHAIN NOR ANY OF HACK-A-CHAIN’S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. HACK-A-CHAIN WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER’S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED “AS IS” AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY

OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT HACK-A-CHAIN'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST HACK-A-CHAIN WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF HACK-A-CHAIN CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST HACK-A-CHAIN WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.
Carmine
FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

SECURITY AUDIT CERTIFICATE

Company Name

We, Hack-a-Chain, Blockchain Specialist Software Development and Audit Company, in this act represented by our Chief Technology Officer, João Antônio Schmidt da Veiga, grant this **Security Audit Certificate** in favor of **Carmine Finance**, recognizing that they underwent the security audit process and corrected all the issues that have been found in their smart contract.

The full security audit report and it's disclaimer can be found in the following link:

https://github.com/hack-a-chain/security-audits

Devoted to enhancing security in the Blockchain Ecosystem and to provide the best quality service for our clients and the community, we sign this Certificate:

João Antônio Schmidt da Veiga
Chief Technology Officer