Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Accreta Markets

Veridise Inc.

February 27, 2026

► **Prepared For:**

Theta Research Labs
www.accreta.markets

► **Prepared By:**

Burak Kadron
Evgeniy Shishkin

► **Contact Us:**

contact@veridise.com

► **Version History:**

Apr. 6, 2026	V3
Apr. 6, 2026	V2
Mar. 31, 2026	V1
Mar. 25, 2026	Initial Draft

Contents

About Veridise

Veridise is an independent security firm founded by academics and security researchers with deep expertise in formal methods, programming languages, and applied cryptography. The firm focuses on high-assurance security for blockchain and cryptographic systems, combining rigorous theory with practical adversarial analysis.

Veridise provides full-stack blockchain security services spanning smart contracts, zero-knowledge circuits and proving systems, consensus and execution clients, cryptographic libraries, and supporting infrastructure. The team routinely analyzes systems at the boundary between protocol design, implementation, and cryptography, where failures are both subtle and high-impact. To date, Veridise has conducted hundreds of security assessments for protocols securing billions of dollars in TVL.

Dynamic Security Veridise treats security as an ongoing engineering discipline rather than a one-time validation exercise. Effective security requires iterative review, continuous feedback, and close collaboration between auditors and system designers. Throughout the engagement, Veridise analysts communicated regularly with the Theta Research Labs team using AuditHub, enabling rapid clarification of design intent, prompt discussion of findings, and efficient validation of fixes.

In parallel with manual review, Veridise maintains active research programs focused on automated and formal analysis of blockchain systems. These efforts have produced tools such as *Vanguard* for static analysis and *Picus* for formal verification, which are available through AuditHub. AuditHub integrates these techniques directly into CI/CD workflows, supporting continuous vulnerability detection and regression analysis beyond the scope of a single audit.

Additional details and case studies are available at `https://audithub.dev/case-studies/`.

Veridise Reports Veridise reports are published in the public audit archive linked below. Only reports obtained from this archive or confirmed directly by a representative of Veridise should be considered official Veridise reports.

`https://veridise.com/audits-archive/`

1 Executive Summary

From Feb. 27, 2026 to Mar. 24, 2026, Theta Research Labs engaged Veridise to conduct a security assessment of the Accreta Markets. The security assessment covered the smart contracts underlying Accreta Markets, a decentralized options protocol that features vault-based RFQ issuance, market-maker quoting, and physically settled options built around Chainlink-based pricing. Veridise conducted the assessment over 34 person-days, with 2 security analysts reviewing the project over 17 days on commit `b05714`.

The review strategy involved a tool-assisted analysis of the program source code performed by Veridise security analysts as well as thorough code review.

Project Summary. The protocol is an on-chain options system centered around a Request-For-Quotation (*RFQ*) workflow. Vaults aggregate user deposits, periodically package that liquidity into option-selling opportunities, and use the *OptionFactory* to request quotes from market makers, who compete to buy or sell option exposure at the best premium. Once an RFQ is settled, the protocol either deploys a new option contract or transfers an existing one, depending on the trade path. The system supports physically settled options, vault-based strategy execution, and oracle-assisted settlement logic based on historical Chainlink pricing.

Main users and their incentives

- **Vault depositors:** deposit assets into vaults in order to earn yield from option premiums and strategy execution without manually managing individual option positions.
- **Market makers:** participate in the RFQ process by quoting premiums or prices, with the goal of acquiring attractive long or short option exposure and monetizing volatility, spread, or inventory advantages.
- **Option buyers and sellers:** hold option positions created either through vault workflows or standalone factory interactions, and use them for speculation, hedging, resale, exercise, or settlement.
- **Vault owner:** manages vault settings and operations, and earns fees from running the vault.
- **Referrers:** can route order flow into the protocol through the referral mechanism and earn a share of protocol economics tied to referred activity.

Main components of the protocol

- **Vaults.** The main entry point for users. Users can deposit assets, request withdrawals of pending deposits, receive shares once deposits are processed, and later withdraw their proportional share of vault assets.
- **OptionFactory.** the protocol's trading and issuance hub. Users can create RFQ requests, submit and reveal RFQ offers, settle quotations, cancel quotations, and in some cases trade existing options rather than minting new ones.
- **Option contracts.** Represent individual option positions. Depending on the role they hold, users can transfer buyer/seller rights, split options into smaller positions, exercise eligible options, explicitly forfeit exercise, reclaim collateral in specific matching-position flows, and close options when both roles converge.

- **Physically settled option layer.** Governs settlement for *Call* and *Put* options where exercise results in an exchange of collateral and delivery assets at the predefined strike-defined conversion ratio.
- **Oracle.** Provides spot and historical pricing, including TWAP-based settlement inputs used to determine whether options are in the money and to support oracle-health-related fallback logic.

Code Assessment. The Accreta Markets developers provided the source code of the Accreta Markets contracts for the code review. The codebase appears largely original and is built around a set of core foundational components that are reused across multiple higher-level products, including different option implementations, vault strategies, oracle adapters, and settlement flows. The architecture is modular, with shared base contracts and utility layers supporting several distinct business workflows rather than a single narrow use case.

The project also stands out positively in terms of engineering maturity: the repository includes strong test coverage across critical paths and edge cases, and the code is accompanied by extensive inline documentation and design-oriented comments, which materially improves reviewability and helps communicate intended behavior and assumptions.

Summary of Issues Detected. The security assessment uncovered 10 issues, 2 of which are assessed to be of medium severity. Specifically, in V-ACC-VUL-001, the protocol accepts collateral price feeds without enforcing the required 8-decimal format, which can cause quotations and vault- created RFQs to become un-settleable at settlement time; in V-ACC-VUL-002, an expired physically settled option with a failing TWAP lookup can indefinitely brick the vault and make liveness depend on buyer cooperation. The Veridise analysts also identified 6 low-severity issues and 1 warnings, such as V-ACC-VUL-003, pending-deposit withdrawals can intermittently fail when a deprecated factory causes the vault to attempt RFQ reissuance; in V-ACC-VUL-004, existing-option RFQ settlement can be griefed by transferring the buyer or seller role just before settlement; in V-ACC-VUL-005, a race between withdrawing pending deposits and RFQ creation can be used to stall vault progression; in V-ACC-VUL-006, the vault assumes asset token addresses are unique without enforcing that invariant; and in V-ACC-VUL-007, TWAP calculation can use a post-target price after a Chainlink phase rollover. Most of the reported issues were fixed by developers; two issues were acknowledged without a fix.

Recommendations. After conducting the assessment of the protocol, the security analysts had a few suggestions to improve the Accreta Markets.

Clarify exact algorithm of Options validation. The `OptionFactory` contract allows individual users to sell existing options to market makers. It is assumed that market makers will validate each option implementation they interact with; however, no reliable validation procedure is specified. It is recommended to clearly define the exact steps market makers should follow to verify the correctness and trustworthiness of option contracts.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Accreta Markets	`b05714`	Solidity	Base

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Feb. 27–Mar. 24, 2026	Manual & Tools	2	34 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	0	0	0
Medium-Severity Issues	2	2	2
Low-Severity Issues	6	6	4
Warning-Severity Issues	1	1	1
Informational-Severity Issues	1	1	1
TOTAL	10	10	8

Table 2.4: Category Breakdown.

Name	Number
Liveness	4
Data Validation	3
Logic Error	2
Maintainability	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Methodology

The security assessment process consists of the following steps:

1. Gather an initial understanding of the protocol's business logic, users, and workflows by reviewing the provided documentation and consulting with the developers.
2. Identify all valuable assets in the protocol.
3. Identify the main workflows for managing these assets.
4. Identify the most significant security risks associated with these assets.
5. Systematically review the codebase for execution paths that could trigger the identified security risks, considering different assumptions.
6. Prioritize one finding over another by assigning a severity level to each.

To improve the depth and efficiency of the code review, analysts utilized the following tools:

- *Static analysis.* To identify potential common vulnerabilities, security analysts leveraged Veridise's custom smart contract analysis tool Vanguard, as well as the open-source tool Slither. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.
- *Fuzzing/Property-based Testing.* Security analysts leveraged fuzz testing to determine if the protocol may deviate from the expected behavior. To do this, the desired behavior of the protocol was formulated as [V] specifications and then tested using Veridise's fuzzing framework OrCa to determine if a violation of the specification can be found.

3.2 Identified Security Risks

After the initial phase of the security assessment was completed, the security analysts generated a list of potential security risks. Security analysts used this list during the code review as a starting point to identify potential attack vectors. A few of these risks include the following:

- Is there a way to take out tokens deposited by users from Vault before an RFQ is created?
- Is there a way to take out tokens deposited by Vault from OptionFactory before the Option is created?
- Is there a way to take out tokens from Option besides the standard ones , like exercise by buyer, withdraw after exercise period or rescue operation?
- Is there a way to trick the best offer algorithm to choose one quote over the other?
- Is there a way to cause OptionFactory to prevent the final settlement of an RFQ?
- Is there a way to mint shares on Vault besides calling `handleOptionEvent()` from the legal OptionFactory contract?
- Is it possible to benefit from already settled or closed RFQ?
- Is it possible to disrupt the Physically Settled Vault-based protocol by passing a malicious `optionBook` implementation?
- Can buyers exploit transient oracle issues to exercise OTM options?
- Can attackers permanently kill the vault via brief oracle outage?
- Does the `settleQuotation()` function behave as intended?

- Can malicious callbacks read an inconsistent state?
- Is there any potential precision loss due to incorrect decimals handling?
- Are there any potential settlement blocking scenarios?
- Are there any potential gas exhaustion opportunities for user-supplied callbacks ?
- Can an attacker benefit from passing a malicious target into `executeExternalCall()` ?

3.3 Scope

The scope of this security assessment is limited to a specific set of source files from the repository, as agreed upon with the Accreta Markets developers:

- `src/libraries/OptionEvents.sol`
- `src/options/BaseOption.sol`
- `src/options/OptionFactory.sol`
- `src/options/PhysicallySettledCallOption.sol`
- `src/options/PhysicallySettledOption.sol`
- `src/options/PhysicallySettledPutOption.sol`
- `src/oracles/HistoricalPriceConsumerV3.sol`
- `src/oracles/HistoricalPriceConsumerV3_FIXEDPRICE.sol`
- `src/oracles/HistoricalPriceConsumerV3_TWAP.sol`
- `src/vaults/BaseVault.sol`
- `src/vaults/FixedStrikeVault.sol`
- `src/vaults/PhysicallySettledVault.sol`
- `src/vaults/utils/DateUtils.sol`
- `src/vaults/utils/EventHandler.sol`
- `src/vaults/utils/IBaseVault.sol`
- `src/vaults/utils/VaultStorage.sol`
- `src/vaults/utils/VaultUtils.sol`

3.4 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

The likelihood of a vulnerability is evaluated according to the Table 3.2.

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

The impact of a vulnerability is evaluated according to the Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Security Review Assumptions

4.1 Operational Assumptions

In addition to assuming that any out-of-scope components behave correctly, Veridise analysts assumed the following properties held when modeling security for Accreta Markets.

- **Protocol owner.** The protocol owner acts in the best interests of the protocol's users and does not perform any actions that would intentionally harm them. In particular, they configure all protocol parameters correctly and use reasonable values.
- **Market makers.** Market makers perform all necessary checks to ensure that the options they interact with originate from trusted parties, such as the vaults of Accreta Markets.
- **Oracle settings.** The protocol relies on well-founded and reputable price feeds, and all related configuration parameters are set correctly.

4.2 Privileged Roles

Roles. This section describes in detail the specific roles present in the system, and the actions each role is trusted to perform. The roles are grouped based on two characteristics: privilege-level and time-sensitivity. *Highly-privileged* roles may have a critical impact on the protocol if compromised, while *limited-authority* roles have a negative, but manageable impact if compromised. Time-sensitive *emergency* roles may be required to perform actions quickly based on real-time monitoring, while *non-emergency* roles perform actions like deployments and configurations which can be planned several hours or days in advance.

During the review, Veridise analysts assumed that the role operators perform their responsibilities as intended. Protocol exploits relying on the below roles acting outside of their privileged scope are considered outside of scope.

- Highly-privileged, emergency roles:
  - **Pause guardian** can pause and unpause the vault alongside the vault owner
- Highly-privileged, non-emergency roles:
  - **OptionFactory owner** can withdraw accumulated fees, change `maxRfqValue` and `baseSplitFee` values, deprecate the factory, acts as the `rescueAddress` for factory created options
  - **Vault owner** can recover failed RFQ settlement manually, execute arbitrary external calls through the vault's owner hook, subject to some blocklist checks, set the pause guardian, withdraw ETH held by the vault, rescue ERC20 tokens from vaults where rescue is implemented, recover a settled RFQ option to the owner in PhysicallySettledVault, update the dynamic IV curve in FixedStrikeVault.

Operational Recommendations. Highly-privileged, non-emergency operations should be operated by a multi-sig contract or decentralized governance system. These operations should be guarded by a timelock to ensure there is enough time for incident response. Highly-privileged, emergency operations should be tested in example scenarios to ensure the role operators are available and ready to respond when necessary.

Full validation of operational security practices is beyond the scope of this review. Users of the protocol should ensure they are confident that the operators of privileged keys are following best practices such as:

- Never storing a protocol key in plaintext, on a regularly used phone, laptop, or device, or relying on a custom solution for key management.
- Using separate keys for each separate function.
- Storing multi-sig keys in a diverse set of key management software/hardware services and geographic locations.
- Enabling 2FA for key management accounts. SMS should *not* be used for 2FA, nor should any account which uses SMS for 2FA. Authentication apps or hardware are preferred.
- Validating that no party has control over multiple multi-sig keys.
- Performing regularly scheduled key rotations for high-frequency operations.
- Securely storing physical, non-digital backups for critical keys.
- Actively monitoring for unexpected invocation of critical operations and/or deployed attack contracts.
- Regularly drilling responses to situations requiring emergency response such as pausing/unpausing.

5 Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 5.1 summarizes the issues discovered:

Table 5.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-ACC-VUL-001	Collateral price feed isn't checked to . . .	Medium	Fixed
V-ACC-VUL-002	Expired option with failing TWAP can . . .	Medium	Fixed
V-ACC-VUL-003	Intermittent DoS in case of deprecated factory	Low	Fixed
V-ACC-VUL-004	Existing-option RFQ settlement can be . . .	Low	Fixed
V-ACC-VUL-005	Potential race between withdrawing and . . .	Low	Acknowledged
V-ACC-VUL-006	Uniqueness of Vault tokens is not established	Low	Fixed
V-ACC-VUL-007	TWAP calculation can use post-target price . . .	Low	Acknowledged
V-ACC-VUL-008	Initial vault shares can be minted for zero . . .	Low	Fixed
V-ACC-VUL-009	Historical price search incorrectly starts . . .	Warning	Fixed
V-ACC-VUL-010	Maintainability issues	Info	Fixed

5.1 Detailed Description of Issues

5.1.1 V-ACC-VUL-001: Collateral price feed isn't checked to provide specific decimals

Severity	Medium	**Commit**	b057140
Type	Data Validation	**Status**	Fixed
Location(s)	`src/options/OptionFactory.sol:617-618`		
Confirmed Fix At	10c09dd		

Description

The protocol assumes that collateral price feeds use 8-decimal precision when calculating settlement fees. However, this requirement is not enforced when an RFQ is created, nor is it validated by the vault when configuring or using an asset price feed. As a result, a quotation can be created successfully with a price feed that exposes a different number of decimals, but later, when settlement attempts to calculate the fee, the transaction reverts because the fee calculation path requires the feed to return exactly 8 decimals.

This creates a mismatch between RFQ admission and RFQ settlement: invalid price feeds are accepted up front, but they make settlement impossible.

The issue also affects vault-created RFQs. The vault relies on `assets[assetIndex].priceFeed` when issuing RFQs, but it does not validate that the configured feed uses the required 8-decimal format. Consequently, a misconfigured vault asset can continuously create RFQs that can never settle.

Impact

Any quotation that relies on a non-8-decimal collateral price feed becomes un-settleable. Once such an RFQ is created, attempts to settle it will revert consistently.

This can lead to:

- stuck RFQs that cannot complete;
- temporary locking of requester and/or winner funds until cancellation paths are used;
- disruption of vault operation if a vault asset is configured with an incompatible price feed, since the vault may keep creating RFQs that are doomed to fail at settlement.

In short, a feed-decimal misconfiguration is promoted from a configuration mistake into a runtime denial-of-service on quotation settlement.

Recommendation

Enforce the feed-decimal requirement at the earliest possible point, rather than only during settlement.

Recommended fixes:

- validate during RFQ creation that the supplied collateral price feed exposes exactly 8 decimals;
- for vaults, validate at configuration or construction time that each `assets[assetIndex].priceFeed` also uses 8 decimals;
- optionally keep the settlement-time check as defense in depth, but do not rely on it as the primary validation layer.

This ensures invalid feeds are rejected immediately and prevents the protocol from creating RFQs that can never settle.

Developers Response

The developers fixed the issue by implementing the recommendation.

5.1.2 V-ACC-VUL-002: Expired option with failing TWAP can indefinitely brick the vault

Severity	Medium	**Commit**	b057140
Type	Liveness	**Status**	Fixed
Location(s)	`src/options/PhysicallySettledOption.sol:312`		
Confirmed Fix At	51eba30		

Description

The vault's main execution paths synchronously process expired options. For physically settled vaults, expired-option handling calls `option.withdraw()`, and that path depends on `isITM(getTWAP())`. If `getTWAP()` reverts for even a single expired option, the withdrawal of that option reverts, which in turn causes expired-option processing to revert.

Because expired-option processing is embedded into the vault's normal flows, one expired option with a failing TWAP can block the vault from progressing. This is not self-healing at the vault level. The protocol instead becomes dependent on the option buyer to manually resolve the position via buyer-controlled paths such as `doNotExercise()` or `exerciseOnOracleFailure()`.

That dependency is unsafe. A buyer may have no incentive to cooperate, especially if the option is economically out-of-the-money, or may intentionally refuse to act. In that case, the expired option remains unresolved and the vault stays stuck indefinitely.

The problem is worsened by the fact that the eventual rescue path is not a proper vault settlement mechanism. After the rescue delay, `rescueERC20()` can move option funds to the configured rescue address rather than resolving them through normal vault accounting, so it does not provide a clean recovery for the vault.

Impact

A single expired option whose TWAP lookup fails can indefinitely brick the vault's core operations.

The practical impact is:

- vault flows that process expired options can revert permanently;
- users may be unable to withdraw or otherwise interact with the vault normally;
- liveness becomes dependent on voluntary buyer action;
- a malicious or indifferent buyer can keep the vault stalled indefinitely;
- the late rescue path does not safely restore normal accounting, since assets can be diverted to the rescue address instead of being resolved through the vault.

This is a protocol-level denial-of-service risk caused by a single expired option and an unavailable oracle/TWAP path.

Recommendation

The vault should not remain dependent on a successful TWAP lookup forever once the exercise window has effectively passed.

Developers Response

The developers fixed the issue by implementing the recommendation.

5.1.3 V-ACC-VUL-003: Intermittent DoS in case of deprecated factory

Severity	Low	**Commit**	b057140
Type	Liveness	**Status**	Fixed
Location(s)	`src/vaults/BaseVault.sol:364`		
Confirmed Fix At	6d170ad		

Description

`BaseVault.withdrawPendingDeposits()` may cancel an active limit-mode RFQ when the withdrawing user pulls funds from the RFQ's collateral asset. After transferring the user's tokens back, the function attempts to immediately reissue the RFQ via `_createOption()`.

The problem is that `_canIssueNewRfq()` does not check whether the `OptionFactory` has been deprecated. If the factory owner has called `deprecateFactory()`, all new RFQ creation attempts revert in `OptionFactory._requestForQuotation()` with `FactoryDeprecated()`.

Impact

Pending deposit withdrawals can become unavailable during a deprecated-factory period whenever the above conditions hold. In practice, users who try to cancel queued deposits may be blocked from retrieving their funds until the vault state changes so that the reissue path is no longer taken.

This is a denial-of-service condition on a user withdrawal path, not merely an inability to create new options. The issue is intermittent because it only manifests while:

- a limit-mode RFQ is active,
- the withdrawing deposits touch that RFQ's collateral asset, and
- the vault is still outside the cutoff window, so it believes reissuing is allowed.

Recommendation

Prevent the vault from attempting RFQ reissuance once the factory has been deprecated. A straightforward fix is:

- expose `deprecationTime` on `OptionFactory` as a public getter; and
- extend `_canIssueNewRfq()` to return `false` when `factory.deprecationTime() != 0`.

This keeps `withdrawPendingDeposits()` on a pure withdrawal path after deprecation: the vault can cancel the stale RFQ, return the user's tokens, and skip `_createOption()` entirely.

Developers Response

The developers fixed the issue by implementing the recommendation.

5.1.4 V-ACC-VUL-004: Existing-option RFQ settlement can be griefed via role transfer

Severity	Low	**Commit**	b057140
Type	Liveness	**Status**	Fixed
Location(s)	`src/options/OptionFactory.sol:1575-1576`		
Confirmed Fix At	245e4be		

Description

For RFQs over an existing option, settlement assumes that the requester still holds the relevant role at the time `settleQuotation()` is executed. This ownership is checked once at RFQ creation in `_requestForQuotation()` and then checked again at settlement in `_settleQuotation()`:

- if `isRequestingLongPosition == false`, the requester must still be the option buyer
- if `isRequestingLongPosition == true`, the requester must still be the option seller

Because option roles are transferable up until settlement, the requester can change the relevant buyer/seller address immediately before `settleQuotation()`. When `_settleQuotation()` re-checks ownership, it detects the mismatch and calls `_abortSettlement(quotationId)` instead of completing the trade.

This lets the requester invalidate the RFQ after market makers have already participated in the commit/reveal flow and after a winner has been selected. While the requester is already allowed to cancel an RFQ explicitly via `cancelQuotation()` before settlement completes, the key difference is that role transfer does not proactively cancel the RFQ when the state changes. The RFQ remains active and appears settleable until a third party spends gas calling `settleQuotation()`, at which point settlement aborts.

Impact

A requester can grief participants in existing-option RFQs by leaving the RFQ live, waiting for offers and winner selection, and then transferring the relevant option role right before settlement. This causes the settlement attempt to fail and the RFQ to be aborted only at the last step.

The practical impact is:

- market makers can waste time and gas participating in RFQs that the requester can invalidate at the final moment;
- whoever calls `settleQuotation()` pays gas for a doomed transaction path;
- the protocol can be left with stale RFQs that remain active even though they are no longer executable.

NOTE: This does not create direct fund theft because `_abortSettlement()` refunds and cancels the RFQ, but it does create a griefing / denial-of-execution vector for existing-option trading.

Recommendation

Ensure that a role transfer on an option backing an active RFQ immediately makes that RFQ non-settleable, rather than waiting for `settleQuotation()` to discover the mismatch.

Developers Response

The developers fixed the issue by implementing the recommendation.

5.1.5 V-ACC-VUL-005: Potential race between withdrawing and RFQ creation

Severity	Low	**Commit**	b057140
Type	Liveness	**Status**	Acknowledged
Location(s)	`src/vaults/BaseVault.sol:364`		
Confirmed Fix At	N/A		

Description

There is a race condition between `createOption()` and `withdrawPendingDeposits()` around the shared `pendingDeposits` queue.

The vault builds new RFQs out of queued pending deposits. At the same time, depositors are allowed to withdraw their pending deposits before they are minted into shares. This means a user who controls a large portion, or even the entirety, of the pending queue can wait for another party to call `createOption()` and front-run that transaction with `withdrawPendingDeposits()`.

If the front-running withdrawal removes the deposits that `createOption()` was relying on, the vault's RFQ creation path may no longer have enough value to issue a valid RFQ. In effect, the attacker can repeatedly empty the queue just before option creation and prevent the protocol from progressing from "pending deposits" into an active RFQ / option lifecycle.

This is not a fund theft issue. It is a liveness issue caused by the fact that queued liquidity can be withdrawn up to the point where someone is trying to consume it for RFQ creation.

Impact

A malicious depositor can grief the vault by repeatedly withdrawing queued deposits just before option creation, especially if they control most or all of the pending queue.

The practical impact is:

- RFQ creation can be prevented or repeatedly reverted;
- queued capital may fail to transition into active strategy positions;
- vault operation can be stalled even though no invariant is violated and no funds are stolen.

The attack is particularly effective when the pending queue is thin and one actor can dominate the deposits being relied upon for the next RFQ.

Recommendation

Reduce or eliminate the ability to withdraw the exact liquidity that is in the process of being used for RFQ creation.

Developers Response

The developers acknowleged the issue with the following comment: "Acknowledged as griefing vector. In practice, mitigated by deployment context: on sequencer-based L2s, transaction ordering is deterministic so sandwich attacks aren't possible. On L1/block-space-auction chains, createOption() can be submitted via MEV-shielded RPCs (Flashbots Protect, etc)."

5.1.6 V-ACC-VUL-006: Uniqueness of Vault tokens is not established

Severity	Low	**Commit**	b057140
Type	Data Validation	**Status**	Fixed
Location(s)	`src/vaults/BaseVault.sol:150-153`		
Confirmed Fix At	6ac6fcf		

Description

The `BaseVault` constructor assumes that each configured asset token address is unique, but it does not enforce this invariant. The vault stores assets in an indexed array and also derives auxiliary mappings and strategy behavior from those asset entries. If the same token address is supplied more than once, the vault ends up with multiple logical asset slots that point to the same ERC20 token. This breaks the implicit one-token-per-asset-index model used throughout the vault logic.

Impact

Using duplicate token addresses can corrupt vault accounting and strategy behavior because the protocol no longer has a one-to-one relationship between asset indexes and actual ERC20 assets.

Potential consequences include:

- incorrect attribution of balances, deposits, or pending deposits across asset indexes;
- incorrect RFQ collateral asset selection or value calculations;
- broken assumptions in withdrawal, minting, and option-tracking flows that rely on asset separation.

Even if this is only reachable through misconfiguration at deployment, the effect is severe because it undermines core accounting assumptions across the vault.

Recommendation

Enforce token-address uniqueness in the constructor.

Developers response

The developers fixed the issue by implementing the recommendation.

5.1.7 V-ACC-VUL-007: TWAP calculation can use post-target price after Chainlink phase rollover

Severity	Low	**Commit**	b057140
Type	Logic Error	**Status**	Acknowledged
Location(s)	`src/oracles/HistoricalPriceConsumerV3_TWAP.sol:44-80`		
Confirmed Fix At	N/A		

Description The protocol relies on `HistoricalPriceConsumerV3_TWAP.calculateTWAP()` to compute expiry-time TWAP values used by options during settlement. For correctness, this TWAP should be built only from oracle rounds that existed at or before the target timestamp.

However, when a Chainlink proxy feed rolls into a new phase after the target timestamp, the current implementation can accidentally use a newer price from that later phase. The function walks backward through rounds to reconstruct the TWAP window, but if the current phase starts entirely after `TARGET_TIMESTAMP`, the scan can reach the phase boundary and step into an invalid proxy round ID `phaseOffset`). That round is not a valid Chainlink round. If this happens before the function has found a valid earlier round in the current phase, the fallback logic uses `latestRoundData()` instead.

As a result, the TWAP can be partially filled using the latest price from the new phase, even though that price did not exist at `TARGET_TIMESTAMP`.

Impact An option can settle using a later-phase price instead of the price history that existed at expiry. This requires the Chainlink proxy feed to roll into a new phase after the option expiry but before settlement. When that happens, any public settlement path that relies on `BaseOption.getTWAP()` can be affected, including exercise and withdrawal flows on expired options.

In affected cases, buyers and sellers may receive incorrect in-the-money results or payout calculations.

Recommendation Stop the backward walk at the first aggregate round in the active phase before `roundID` is decremented into `phaseOffset`, and source the frozen price from the last valid round before the phase change instead of from `latestRoundData()` when no current-phase round precedes `TARGET_TIMESTAMP`.

Enforce this guard inside `HistoricalPriceConsumerV3_TWAP.calculateTWAP` immediately before `roundID -= 1` and inside the phase-boundary fallback branch so the contract never queries `getRoundData(phaseOffset)` or substitutes a post-target latest price for the missing history.

Developer Response The developers acknowledged the issue, but decided not to provide the fix with the following comment: "this issue requires 1) unannounced phase migration + 2) extreme price move over the 30 min TWAP + 3) delayed settlement to exploit e.g. all parties and watchers fail to settle - acknowledged but infeasible."

5.1.8 V-ACC-VUL-008: Initial vault shares can be minted for zero assets with non-standard tokens

Severity	Low	**Commit**	b057140
Type	Data Validation	**Status**	Fixed
Location(s)	`src/vaults/BaseVault.sol:275-276`		
Confirmed Fix At	c512995		

Description

`BaseVault.depositOnBehalf()` checks the deposit size using the user-supplied amount, but the vault's actual accounting is based on `actualReceived`, which is calculated from the token balance change after the transfer. This creates a mismatch for non-standard tokens. In particular, if a token's transfer behavior or rebasing mechanics cause the vault's balance increase to be smaller than amount, the vault will still validate the deposit using the larger requested value. In the extreme case, `actualReceived` can theoretically become 0 even though amount > 0.

This matters most on the initial deposit path. When `totalSupply() == 0`, the vault mints the initial share supply as long as the value check based on amount passes. It does not require that `actualReceived` is non-zero. As a result, the first depositor can theoretically receive the initial shares even if the vault received no assets.

Impact

A vault can mint initial shares against zero received assets if it is used with a non-standard token whose balance change does not match the requested transfer amount. This is primarily a correctness and accounting issue. It breaks the assumption that minted shares are backed by assets actually received by the vault. In the worst case, it lets the first depositor establish vault ownership without contributing any real assets.

The issue is most relevant for rebasing or otherwise unusual ERC20 tokens. It is unlikely to affect standard tokens, but the code explicitly tries to support tokens where `actualReceived` may differ from amount, so this edge case should be handled safely.

Recommendation

The vault should validate deposits using `actualReceived`, not the requested amount.

In particular:

- revert if `actualReceived == 0`;
- use `actualReceived` when checking minimum deposit value;
- ensure the initial mint path only mints shares when the vault has actually received assets.

This keeps vault accounting aligned with the real token balance change and prevents initial shares from being minted for zero assets.

Developers Response

The developers fixed the issue by implementing the recommendation.

5.1.9 V-ACC-VUL-009: Historical price search incorrectly starts from Chainlink round 0

Severity	Warning	**Commit**	b057140
Type	Logic Error	**Status**	Fixed
Location(s)	`src/oracles/HistoricalPriceConsumerV3.sol:181`		
Confirmed Fix At	071d564		

Description

The historical price lookup logic assumes that round 0 is a valid lower bound when searching within a Chainlink phase. In `HistoricalPriceConsumerV3`, the search path calls `findBlockSamePhase(timeStamp, phaseOffset, 0, end)`, which means the binary search may query the packed round ID corresponding to `aggregatorRoundId = 0`.

This is not aligned with standard Chainlink round numbering. For normal Chainlink feeds, valid rounds start at `1`, while `0` is treated as a sentinel / invalid round rather than a real data round. As a result, the historical lookup code may probe a non-existent round at the beginning of the search range.

Impact

Using `0` as the starting round can cause historical price lookup to fail unexpectedly when the search reaches the lower edge of a phase.

The practical impact is:

- historical price queries may revert for valid timestamps near the start of a phase;
- TWAP calculations that depend on this historical lookup can fail;
- any protocol path that relies on those TWAP results may become unexecutable.

Because the issue sits in shared historical price infrastructure, its effect can propagate into option settlement and vault workflows that depend on historical oracle reads.

Recommendation

Replace `start = 0` with `start = 1` for standard Chainlink feeds.

Developers Response

The developers fixed the issue by implementing the recommendation.

5.1.10 V-ACC-VUL-010: Maintainability issues

Severity	Info	**Commit**	b057140
Type	Maintainability	**Status**	Fixed
Location(s)	`src/` ▶ `options/BaseOption.sol:266-269, 659-661` ▶ `oracles/` • `HistoricalPriceConsumerV3.sol:177-181` • `HistoricalPriceConsumerV3_TWAP.sol:33-36, 58-59` ▶ `vaults/` • `BaseVault.sol:155-169, 561-566, 814-819` • `utils/EventHandler.sol:40-52`		
Confirmed Fix At	1e8cae4		

Description Several core modules contain stale comments, unused variables, redundant checks, and repeated magic values.

- `BaseOption.NOTIFY_GAS_LIMIT` still mentions budgeting for `EVENT_CREATION`, even though `EventHandler.handleEvent()` has no `EVENT_CREATION` branch and ignores unknown events.
- `HistoricalPriceConsumerV3` and `HistoricalPriceConsumerV3_TWAP` repeat the phase-width literal `2 ** 64` inline instead of naming the Chainlink round-width constant once.
- `BaseVault` computes `maxDurationDays` and then discards it after the RFQ frequency check was removed.
- `_issueNewRfq` assigns the return value of `_OptionFactory_getQuotation(activeRfqId)` back into `params` even though that value is never consumed.
- `_handleActiveRfq` rechecks `block.timestamp > params.offerEndTimestamp + REVEAL_WINDOW` inside a branch that already established the same predicate.
- `BaseOption.split` still relies on an unchecked ERC20 `approve` return value plus an external comment to encode its fresh-allowance assumption.

Impact The code becomes harder to read and reason about correctly. A reviewer or future maintainer must spend extra effort separating live behavior from old leftovers. In sensitive areas such as callback gas limits, RFQ recovery, and Chainlink phase handling, that increases the chance of future mistakes.

Recommendation Clean up `BaseOption`, `BaseVault`, `HistoricalPriceConsumerV3`, and `HistoricalPriceConsumerV3_TWAP` so comments, constants, and local variables encode only behavior that still executes.

1. Rewrite or remove the `NOTIFY_GAS_LIMIT` comment in `BaseOption` so it describes the event handlers that the vault path actually processes today.
2. Introduce a shared named constant for the Chainlink phase width and use it in both historical-price consumers instead of repeating `2 ** 64` inline.
3. Delete `maxDurationDays`, remove the dead `(params,) = _OptionFactory_getQuotation(activeRfqId)` assignment unless it is replaced with an explicit validation helper, and collapse the redundant timestamp comparison in `_handleActiveRfq`.

4. Route the ERC20 allowance update through a helper that encodes the intended `0 -> X` approval assumption, either by checking the boolean return value or by using the existing `SafeERC20` utilities consistently.

Developer Response The developers fixed the issue by addressing most of the raised concerns.

6 Fuzz Testing

6.1 Methodology

One of the goals of the security assessment was to test fuzz Accreta Markets to evaluate its functional correctness, i.e, whether the implementation deviates from the intended behavior.

The Veridise security analysts used the OrCa tool to fuzz the project. The analysts captured the intended behavior of the system by writing invariants–logical formulas which should hold after each transaction. The invariants were encoded as statements in the [V] specification language and passed to the OrCa tool, together with the project program code.

The security analysts have written deployment scripts to set up the fuzzing environment with contracts in scope and some mock contracts provided in the Foundry testing suite of the project (namely Chainlink price feed and `ERC20` contracts). To be able to call certain functions successfully, guide the fuzzing process, and access certain variables easily, the analysts have written OrCa hints and helper contracts. The analysts have used OrCa's function blacklisting to prevent fuzzing certain functions such as `FixedStrikeStrategyVault.createOption` and `OptionFactory.requestForQuotation` to be called during fuzzing to limit the size of the fuzzing environment and prevent out of memory errors on OrCa.

6.2 Properties Fuzzed

Table 6.1 describes the fuzz-tested invariants. The second column describes the invariant informally in English, and the third shows the total amount of compute time spent fuzzing this property. The last column indicates the number of bugs identified when fuzzing the invariant.

The properties include correctness specifications on `BaseOption`, `FixedStrikeStrategyVault`, `HistoricalPriceConsumerV3_TWAP`, `OptionFactory`, and `PhysicallySettledOption` contracts.

The Veridise team devoted a total of 3 compute-hours to fuzzing this protocol, identifying a total of 2 bugs.

Table 6.1: Invariants Fuzzed.

Specification	Invariant	Minutes Fuzzed	Bugs Found
V-ACC-SPEC-001	BaseOption.01: Option ERC20 rescue is only . . .	90	0
V-ACC-SPEC-002	BaseOption.02: Splitting an option must . . .	90	0
V-ACC-SPEC-003	BaseOption.03: Buyer and seller role . . .	90	0
V-ACC-SPEC-004	FixedStrikeStrategyVault.01: . . .	90	0
V-ACC-SPEC-005	FixedStrikeStrategyVault.02: Vault assets . . .	90	0
V-ACC-SPEC-006	FixedStrikeStrategyVault.03: Only the owner . . .	90	0
V-ACC-SPEC-007	FixedStrikeStrategyVault.04: The vault must . . .	90	0
V-ACC-SPEC-008	HistoricalPriceConsumerV3_TWAP.01: . . .	90	1
V-ACC-SPEC-009	OptionFactory.01: Cancelling an RFQ tied to . . .	90	0
V-ACC-SPEC-010	OptionFactory.02: Only the requester may . . .	90	0
V-ACC-SPEC-011	OptionFactory.03: Only the factory owner . . .	90	0
V-ACC-SPEC-012	OptionFactory.04: Tracked pending fees . . .	90	0
V-ACC-SPEC-013	OptionFactory.05: The first revealed winning . . .	90	0
V-ACC-SPEC-014	OptionFactory.06: Once a winner exists, later . . .	90	0
V-ACC-SPEC-015	OptionFactory.07: A non-requester cannot . . .	90	0
V-ACC-SPEC-016	OptionFactory.08: A successful reveal must . . .	90	0
V-ACC-SPEC-017	OptionFactory.09: Settling a quotation . . .	90	0
V-ACC-SPEC-018	PhysicallySettledOption.01: simulatePayout . . .	90	1
V-ACC-SPEC-019	PhysicallySettledOption.02: Core option . . .	90	0
V-ACC-SPEC-020	PhysicallySettledOption.03: An initialized . . .	90	0

6.3 Detailed Description of Fuzzed Specifications

6.3.1 V-ACC-SPEC-001: BaseOption.01: Option ERC20 rescue is only allowed for the rescue address after the post-expiry delay.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to `BaseOption.rescueERC20` in `/src/options/BaseOption.sol`.

Natural Language Rescuing ERC20s from an option is a tightly constrained recovery path. The caller must be the configured `rescueAddress`, and if the option has a finite expiry, the rescue can only happen at least one day after expiry.

Formal The [V] spec is described as below:

```
vars: BaseOption option, IERC20 token
spec: []!finished(
    option.rescueERC20(token),
    sender != old(option.rescueAddress())
    || (option.expiryTimestamp() != MAX_UINT256 && timestamp < option.expiryTimestamp
    () + DAY)
)
```

Example No example of violation of this specification has been found while fuzzing.

6.3.2 V-ACC-SPEC-002: BaseOption.02: Splitting an option must preserve the expected parent-child accounting relationship.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to `BaseOption.split` in `/src/options/BaseOption.sol` and to the newly created child option returned by that call.

Natural Language When an option is split, the parent and child states should remain consistent with the pre-split collateral, contract count, and split generation. The operation must conserve the expected accounting relationship across both option instances.

Formal The [V] spec is described as below:

```
vars: BaseOption option, OrCaOptionSpecView viewHelper
spec: []!finished(
    option.split(splitCollateralAmount),
    !viewHelper.splitStateMatches(
        option, ret, old(option.collateralAmount()), old(option.numContracts()), old(
    option.splitGeneration())
    )
)
```

Example No example of violation of this specification has been found while fuzzing.

6.3.3 V-ACC-SPEC-003: BaseOption.03: Buyer and seller role transfers require authorized callers.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to `BaseOption.transfer` in `/src/options/BaseOption.sol` for both buyer-side and seller-side role transfers.

Natural Language An option role transfer is only valid if initiated by the factory, the current role holder, or an approved delegate for that role. Any other caller must be prevented from moving buyer or seller rights.

Formal The [V] spec is described as below:

```
vars: BaseOption option
spec: []!finished(
    option.transfer(isBuyer, target),
    sender != old(option.factory())
        && (
            (isBuyer && sender != old(option.buyer()) && !old(option.buyerAllowance(
    old(option.buyer()), sender)))
                || (
                    !isBuyer && sender != old(option.seller())
                        && !old(option.sellerAllowance(old(option.seller()), sender))
                )
        )
)
```

Example No example of violation of this specification has been found while fuzzing.

6.3.4 V-ACC-SPEC-004: FixedStrikeStrategyVault.01: depositOnBehalf must queue the intended beneficiary, asset index, and amount.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to `FixedStrikeStrategyVault.depositOnBehalf` in `/src/vaults/FixedStrikeVault.sol` when deposits are queued rather than minted immediately.

Natural Language If deposits are being queued, a deposit made on behalf of someone else must record the beneficiary, asset index, and deposited amount in the pending queue. Relayers and wrappers must not cause the queue to attribute the deposit to the wrong user or a zero amount.

Formal The [V] spec is described as below:

```
vars: FixedStrikeStrategyVault vault, OrCaVaultSpecView viewHelper
spec: []!finished(
    vault.depositOnBehalf(beneficiary, amount, assetIndex),
    old(vault.totalSupply()) > 0
        && (
            viewHelper.lastPendingDepositUser(vault) != beneficiary
                || viewHelper.lastPendingDepositAssetIndex(vault) != assetIndex
                || viewHelper.lastPendingDepositAmount(vault) = 0
        )
)
```

Example No example of violation of this specification has been found while fuzzing.

6.3.5 V-ACC-SPEC-005: FixedStrikeStrategyVault.02: Vault assets may only be rescued while the vault is in recovery mode.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to `FixedStrikeStrategyVault.rescueERC20` in `/src/vaults/FixedStrikeVault.sol` for tokens that are recognized as vault assets.

Natural Language Rescuing tracked vault assets is intended as a recovery-only escape hatch. During normal operation, even the owner must not be able to extract vault assets through the rescue function.

Formal The [V] spec is described as below:

```
vars: FixedStrikeStrategyVault vault, OrCaVaultSpecView viewHelper
spec: []!finished(
    vault.rescueERC20(token),
    !old(vault.isInRecoveryMode()) && viewHelper.isVaultAsset(vault, token)
)
```

Example No example of violation of this specification has been found while fuzzing.

6.3.6 V-ACC-SPEC-006: FixedStrikeStrategyVault.03: Only the owner or pause guardian may toggle the vault pause state.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to `FixedStrikeStrategyVault.setPause` in `/src/vaults/FixedStrikeVault.sol`.

Natural Language Pausing and unpausing the vault is a privileged liveness control. Only the vault owner or the configured `pauseGuardian` should be able to change that state.

Formal The [V] spec is described as below:

```
vars: FixedStrikeStrategyVault vault
spec: []!finished(vault.setPause(pausedValue), sender != old(vault.owner()) && sender
     != old(vault.pauseGuardian()))
```

Example No example of violation of this specification has been found while fuzzing.

6.3.7 V-ACC-SPEC-007: FixedStrikeStrategyVault.04: The vault must block external calls to sensitive or forbidden targets.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to `FixedStrikeStrategyVault.executeExternalCall` in `/src/vaults/FixedStrikeVault.sol`.

Natural Language The vault's owner-controlled arbitrary-call mechanism must not be able to target blocked contracts such as vault internals, protected infrastructure, or sensitive asset endpoints. If a target is classified as blocked, the external call must not successfully finish.

Formal The [V] spec is described as below:

```
vars: FixedStrikeStrategyVault vault, OrCaVaultSpecView viewHelper
spec: []!finished(
    vault.executeExternalCall(target, data),
    viewHelper.isBlockedExecuteExternalCallTarget(vault, target)
)
```

Example No example of violation of this specification has been found while fuzzing.

6.3.8 V-ACC-SPEC-008: HistoricalPriceConsumerV3_TWAP.01: TWAP computation must use the correct historical window instead of leaking in the latest round across a phase boundary.

Minutes Fuzzed	90	Bugs Found	1

Scope This spec applies to TWAP calculation logic in `HistoricalPriceConsumerV3_TWAP` in `src/oracles/HistoricalPriceConsumerV3_TWAP.sol`. The spec below exercises that call through `OrCaPriceFeedHelper` in `/src/mocks/OrCaPriceFeedHelper.sol` . We have added a helper contract `OrCaPriceFeedHelper` as OrCa does not fuzz pure and view functions and we did not want to enable fuzzing of all pure and view functions as that would make fuzzing very inefficient.

Natural Language For a historical target timestamp, the TWAP should be derived from round data covering that target window. It must not accidentally anchor to the latest Chainlink round just because the feed crossed a phase boundary.

Formal The [V] spec is described as below:

```
vars: OrCaPriceFeedHelper priceHelper
spec: []!finished(
    priceHelper.calculateTWAP(twapOracle, priceFeed, targetTimestamp, period),
    targetTimestamp < priceHelper.extractLatestRoundData(priceFeed)[1] &&
    priceHelper.getPriceAfterTimestamp(priceFeed, targetTimestamp) != ret &&
    priceHelper.extractLatestRoundData(priceFeed)[0] = ret
)
```

Example OrCa finds a counterexample by calling calculateTWAP with the parameters below where that timestamp corresponds to a different price but TWAP calculation reverts to the price of the latest round data.

The counterexample can be achieved by calling `calculateTWAP` with the following parameters:

```
OrCaPriceFeedHelper(0x2279).calculateTWAP{sender: 0xf39f}(twap: 0x5fc8, priceFeed: 0
    x6101, TARGET_TIMESTAMP: 1757783214, TWAP_PERIOD: 60);
```

6.3.9 V-ACC-SPEC-009: OptionFactory.01: Cancelling an RFQ tied to an existing option must make that quotation inactive.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to `OptionFactory.cancelQuotation` in `/src/options/OptionFactory.sol` when the quotation references a nonzero `existingOptionAddress`.

Natural Language If a quotation is attached to an already-deployed option, then a successful cancellation must fully deactivate that quotation. The system should never leave an existing-option RFQ marked active after cancellation finishes.

Formal The [V] spec is described as below:

```
vars: OptionFactory factory
spec: []!finished(
    factory.cancelQuotation(qid),
    factory.quotations(qid)[0].existingOptionAddress != nulladdr && factory.
    quotations(qid)[1].isActive
)
```

Example No example of violation of this specification has been found while fuzzing.

6.3.10 V-ACC-SPEC-010: OptionFactory.02: Only the requester may cancel a quotation and a genuinely stuck quotation may be cancelled by a non-requester.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to third-party calls to `OptionFactory.cancelQuotation` in `/src/options/OptionFactory.sol` for live quotations.

Natural Language An outside caller must not be able to cancel a quotation while it is still progressing normally. A non-requester cancellation is only acceptable after the quotation is stuck beyond the offer and reveal windows and is not already in a winner-selected or limit-order path.

Formal The [V] spec is described as below:

```
vars: OptionFactory factory
spec: []!finished(
    factory.cancelQuotation(qid),
    qid < factory.getQuotationCount()
    && sender != old(factory.quotations(qid)[0].requester)
    && (
        timestamp < old(factory.quotations(qid)[0].offerEndTimestamp) + factory.
    REVEAL_WINDOW()
            || old(factory.quotations(qid)[0].convertToLimitOrder)
            || old(factory.quotations(qid)[1].currentWinner != nulladdr)
    )
)
```

Example No example of violation of this specification has been found while fuzzing.

6.3.11 V-ACC-SPEC-011: OptionFactory.03: Only the factory owner may reduce pending fee balances.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to any `OptionFactory` execution path in `/src/options/OptionFactory.sol` that decreases `pendingFees`.

Natural Language If the tracked pending fees for a token go down, that reduction must have been initiated by the factory owner. No other caller should be able to drain or decrement the fee ledger.

Formal The [V] spec is described as below:

```
vars: OptionFactory factory, IERC20 feeToken
spec: []!finished(factory.*, sender != factory.owner() && factory.pendingFees(
    feeToken) < old(factory.pendingFees(feeToken)))
```

Example No example of violation of this specification has been found while fuzzing.

6.3.12 V-ACC-SPEC-012: OptionFactory.04: Tracked pending fees must never exceed the factory's actual token balance.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to fee accounting in `OptionFactory` in `/src/options/OptionFactory.sol`, across all state transitions and fee tokens.

Natural Language For every fee token, the factory must hold at least as many tokens as it records in `pendingFees`. Internal accounting should never promise more withdrawable fees than the contract actually owns.

Formal The [V] spec is described as below:

```
vars: OptionFactory factory, IERC20 feeToken
spec: []!finished(factory.*,
    feeToken.balanceOf(address(factory)) < factory.pendingFees(feeToken)
)
```

Example No example of violation of this specification has been found while fuzzing.

6.3.13 V-ACC-SPEC-013: OptionFactory.05: The first revealed winning offer must not be worse than the reserve or current anchor price.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to `OptionFactory.revealOffer` in `/src/options/OptionFactory.sol` when the reveal establishes the first winner for a quotation.

Natural Language Before any winner exists, the first acceptable revealed offer must meet the requester's pricing direction. For long-position requests, the offer cannot be above the current best price or reserve; for the opposite direction, it cannot be below that anchor.

Formal The [V] spec is described as below:

```
vars: OptionFactory factory, OrCaOptionFactorySpecView viewHelper
spec: []!finished(
    factory.revealOffer(qid, offerAmount, nonce, offeror),
    !old(viewHelper.hasWinner(factory, qid)) && sender != viewHelper.requester(
    factory, qid) && old(viewHelper.currentBestPriceOrReserve(factory, qid)) > 0 &&
    ((old(viewHelper.isRequestingLongPosition(factory, qid)) && offerAmount > old(
    viewHelper.currentBestPriceOrReserve(factory, qid))) || (!old(viewHelper.
    isRequestingLongPosition(factory, qid)) && offerAmount < old(viewHelper.
    currentBestPriceOrReserve(factory, qid)))))
```

Example No example of violation of this specification has been found while fuzzing.

6.3.14 V-ACC-SPEC-014: OptionFactory.06: Once a winner exists, later revealed offers must strictly improve the best price.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to `OptionFactory.revealOffer` in `/src/options/OptionFactory.sol` after a quotation already has a current winner.

Natural Language After the auction has a winner, the best price must move only in the favorable direction. A later reveal cannot match or worsen the existing best offer and still be accepted as progress.

Formal The [V] spec is described as below:

```
vars: OptionFactory factory, OrCaOptionFactorySpecView viewHelper
spec: []!finished(factory.revealOffer(qid, offerAmount, nonce, offeror), old(
    viewHelper.hasWinner(factory, qid)) && ((old(viewHelper.isRequestingLongPosition(
    factory, qid)) && offerAmount >= old(viewHelper.currentBestPriceOrReserve(factory
    , qid))) || (!old(viewHelper.isRequestingLongPosition(factory, qid)) &&
    offerAmount <= old(viewHelper.currentBestPriceOrReserve(factory, qid)))))
```

Example No example of violation of this specification has been found while fuzzing.

6.3.15 V-ACC-SPEC-015: OptionFactory.07: A non-requester cannot reveal an offer before the offer window has ended.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to `OptionFactory.revealOffer` in `/src/options/OptionFactory.sol` for callers other than the quotation requester.

Natural Language Outside participants must respect the commit-then-reveal schedule. If the caller is not the requester, revealing before `offerEndTimestamp` must not successfully finish.

Formal The [V] spec is described as below:

```
vars: OptionFactory factory
spec: []!finished(
    factory.revealOffer(qid, offerAmount, nonce, offeror),
    qid < factory.getQuotationCount()
    && sender != old(factory.quotations(qid)[0].requester)
    && timestamp <= old(factory.quotations(qid)[0].offerEndTimestamp)
)
```

Example No example of violation of this specification has been found while fuzzing.

6.3.16 V-ACC-SPEC-016: OptionFactory.08: A successful reveal must update both the current winner and the tracked best price.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to successful `OptionFactory.revealOffer` executions in `/src/options/OptionFactory.sol`.

Natural Language When an offer reveal succeeds, the quotation state must reflect that exact offer. The winner should become the revealed `offeror`, and the best tracked price should become the revealed `offerAmount`.

Formal The [V] spec is described as below:

```
vars: OptionFactory factory, OrCaOptionFactorySpecView viewHelper
spec: []!finished(factory.revealOffer(qid, offerAmount, nonce, offeror), viewHelper.
    currentWinner(factory, qid) != offeror || viewHelper.currentBestPriceOrReserve(
    factory, qid) != offerAmount)
```

Example No example of violation of this specification has been found while fuzzing.

6.3.17 V-ACC-SPEC-017: OptionFactory.09: Settling a quotation without a winner is only valid in limit-order mode.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to `OptionFactory.settleQuotation` in `/src/options/OptionFactory.sol` when no winner has been recorded.

Natural Language A normal auction-style RFQ should not settle down the no-winner path unless it was explicitly configured to convert into a limit order. If there is no winner and limit mode is off, settlement must not successfully finish.

Formal The [V] spec is described as below:

```
vars: OptionFactory factory, OrCaOptionFactorySpecView viewHelper
spec: []!finished(
    factory.settleQuotation(qid),
    !old(viewHelper.hasWinner(factory, qid)) && !old(viewHelper.convertToLimitOrder(
    factory, qid))
)
```

Example No example of violation of this specification has been found while fuzzing.

6.3.18 V-ACC-SPEC-018: PhysicallySettledOption.01: simulatePayout must match calculatePayout for the same option state and input price.

Minutes Fuzzed	90	Bugs Found	1

Scope This spec applies to payout logic in `PhysicallySettledOption` in `/src/options/PhysicallySettledOption.sol`.

Natural Language The option's simulation helper and its actual payout calculation should agree for identical inputs. Any divergence means the modeling function no longer describes the real payout path.

Formal The [V] spec is described as below:

```
vars: PhysicallySettledOption option, uint256 price
spec: []!finished(
    option.*,
    option.simulatePayout(price, option.getStrikes(), option.numContracts) != option.
    calculatePayout(price)
)
```

Example OrCa finds a counterexample to the specification with the strike and numContracts values set as below. We have added a helper function `updateParams` to `PhysicallySettledCallOption` and `PhysicallySettledPutOption` to be able to fuzz the `strike` and `numContracts` parameters without re-initializing the option where `updateParams` calculates the correct `collateralAmount` given the values and updates `strike`, `numContracts`, and `collateralAmount` fields of the option.

The developers have acknowledged the mismatch between `simulatePayout` and `calculatePayout` functions but said the behavior was intended as both functions are used in different parts of the code and the difference is based on rounding errors when strike price is not divisible by `1e8`.

The counterexample can be achieved by calling the helper function `updateParams` with the following parameters:

```
PhysicallySettledPutOption(0x0165).updateParams{sender: 0x15d3}(_strike: 7.20576e+16,
     _numContracts: 6.2771e+57);
```

6.3.19 V-ACC-SPEC-019: PhysicallySettledOption.02: Core option configuration must remain immutable after initialization.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to initialized `PhysicallySettledOption` instances in `/src/options/PhysicallySettledOption.sol`.

Natural Language Once an option has been initialized, its identity and settlement-critical configuration should be fixed. Fields such as the params hash, tokens, oracle settings, factory linkage, creator, and parent linkage must not change afterward.

Formal The [V] spec is described as below:

```
vars: PhysicallySettledOption option
spec: []!finished(option.*,
    old(option.collateralToken()) != nulladdr &&
    (
        option.paramsHash() != old(option.paramsHash()) ||
        option.collateralToken() != old(option.collateralToken()) ||
        option.chainlinkPriceFeed() != old(option.chainlinkPriceFeed()) ||
        option.twapPeriod() != old(option.twapPeriod()) ||
        option.rescueAddress() != old(option.rescueAddress()) ||
        option.factory() != old(option.factory()) ||
        option.historicalTWAPConsumer() != old(option.historicalTWAPConsumer()) ||
        option.creator() != old(option.creator()) ||
        option.deliveryCollateral() != old(option.deliveryCollateral()) ||
        option.optionParent() != old(option.optionParent())
    )
)
```

Example No example of violation of this specification has been found while fuzzing.

6.3.20 V-ACC-SPEC-020: PhysicallySettledOption.03: An initialized option must never be initialized a second time.

Minutes Fuzzed	90	Bugs Found	0

Scope This spec applies to `PhysicallySettledOption.initialize` in `/src/options/PhysicallySettledOption.sol` after the option has already been set up.

Natural Language Clone-based option instances rely on one-time initialization. If `collateralToken` is already nonzero, any later `initialize` call must fail rather than overwriting state.

Formal The [V] spec is described as below:

```
vars: PhysicallySettledOption option
spec: []!finished(option.initialize(p), old(option.collateralToken()) != nulladdr)
```

Example No example of violation of this specification has been found while fuzzing.