



Security Review For Convallax



Collaborative Audit Prepared For:
Lead Security Expert(s):

Convallax

0xeix

oot2k

Date Audited:

June 19 - June 21, 2026

Introduction

Convallax enables users to trade options on prediction markets. Prediction markets today are limited to binary Yes or No bets. Options, or calls and puts on Yes and No underlying shares, enable leverage, volatility trading, and hedging of Yes/No exposure - all critical use cases for the growing prediction market economy.

Prediction markets are binary options. But since Yes/No can be traded, they can become the underlying of an option themselves. On Convallax, users can trade calls and puts on these probabilities. Buy a call to bet that the Yes probability of the prediction market will be above a certain strike at a certain expiry, or buy a put to bet that the Yes probability will be below the strike at expiry.

Leverage on prediction markets is the retail use case that Convallax is spear-heading. Options are the best solution to leverage on prediction markets, because perps and margin suffer from price jumps that wipe out collateral or force positions to be closed early. But with options, only pay a small premium, and earn a large payout if you're right, rather than putting up the whole cost of the Yes/No share. With options, you get levered exposure to the direction you're betting on.

Options also provide an effective way for users to earn yield on the Yes/No positions they hold, via covered calls or cash-secured puts. By writing options, users earn premium upfront, in exchange for giving up some of the upside of the underlying - though even this tradeoff is limited because the underlying itself is bounded.

Lastly, as options on prediction markets mature, the world will have a way to not just price the implied probability of an event, but also the uncertainty of where the market thinks the probability will be at various times in the future. A two-dimensional belief surface on probability and time will be a useful tool for society to price not just the likelihood of an event itself, but the uncertainty around it and the probability's path from now up until the event itself.

Convallax is a fully-collateralized RFQ system: users select an underlying market, long or short, an option type (call or put), a strike between 0 and 1, and an expiry. The user then sends a request-for-quote (RFQ) which is received by market-makers integrated onto the platform. Market-makers run off-chain pricing models to generate a quote, and the best quote is returned to the user. When a trade is accepted, the premium is transferred from the buyer's wallet to the writer's, collateral is deposited from the writer to a vault, and option tokens are minted to the buyer's wallet, in one atomic transaction.

Convallax is building *the* exchange for options on prediction markets, pioneering a novel primitive that will dwarf prediction markets themselves just like derivatives have done in TradFi and crypto. As any spot market matures, the options and derivatives layer dwarfs the underlying in trading activity, and that stage is fast approaching for prediction markets. As prediction markets continue to mature, Convallax is positioned to become the primary options layer on top for leverage, hedging, risk management, and volatility.

Scope

Repository: kushal613/Convallax

Audited Commit: a3236d6ba28610fab930dc75c5a1b57c74ab0ad1

Final Commit: a71fbbee4455f63d4c1fadfa9b5a52ab5bae43c1

Files:

- contracts/src/ConvallaxCore.sol
- contracts/src/ConvallaxOptionToken.sol
- contracts/src/ConvallaxRFQSettlement.sol
- contracts/src/interfaces/IConvallaxCore.sol
- contracts/src/interfaces/IConvallaxRFQSettlement.sol
- contracts/src/libraries/OptionMath.sol
- contracts/src/MockUSDC.sol

Final Commit Hash

[a71fbbee4455f63d4c1fadfa9b5a52ab5bae43c1](#)

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
2	0	5

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Writer claims permanently strand collateral and depend on claim order [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-06-convallax-june-19th-2026/issues/8>

Summary

Writer collateral claims are calculated using the decreasing `writerPoolRemaining` but the original `totalCollateral` denominator. This makes payouts claim-order dependent and permanently strands part of the writer pool.

Vulnerability Detail

The first writer receives their expected proportional share. However, after their payout is deducted, subsequent writers calculate their entitlement against the smaller remaining pool while still dividing by the original total collateral:

<https://github.com/sherlock-audit/2026-06-convallax-june-19th-2026/blob/main/Convallax/contracts/src/libraries/OptionMath.sol#L132-139>

```
function writerClaim(
    uint256 writerCollateral,
    uint256 writerPoolRemaining,
    uint256 totalCollateral
) internal pure returns (uint256) {
    if (totalCollateral == 0) return 0;
    return (writerCollateral * writerPoolRemaining) / totalCollateral;
}
```

Imagine two writers deposit collateral into one option series:

```
Alice deposits 3 USDC
Bob deposits 2 USDC
Total collateral is 5 USDC
```

After settlement, holders are owed 0.90 USDC, so writers should split the remaining 4.10 USDC.

Correct result will be:

```
Alice owns 60%, so she should get 2.46 USDC
Bob owns 40%, so he should get 1.64 USDC
Total paid back: 4.10 USDC
```

But the contract does this:

```
return (writerCollateral * writerPoolRemaining) / totalCollateral;
```

Alice claims first:

```
3 * 4.10 / 5 = 2.46 USDC
```

Remaining writer pool becomes 1.64 USDC. Bob claims second:

```
2 * 1.64 / 5 = 0.656 USDC
```

The protocol's specification also confirms that writers has to reclaim unspent collateral pro-rata.

Impact

Because `writerPoolRemaining` shrinks after each claim but `totalCollateral` stays fixed, later writers are underpaid and collateral can remain stuck.

Tool Used

Manual Review

Recommendation

Either calculate each writer's payout from an immutable settlement-time snapshot, or reduce both the remaining pool and remaining collateral denominator after every claim.

Issue H-2: Mint rounding can permanently block settlement [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-06-convallax-june-19th-2026/issues/9>

Summary

Collateral is rounded down separately for each mint, while holder liability is calculated using the aggregate option supply. Multiple small mints can therefore create a collateral deficit that causes settlement to revert permanently.

Vulnerability Detail

The protocol calculates collateral for each mint using floor division:

<https://github.com/sherlock-audit/2026-06-convallax-june-19th-2026/blob/main/Convallax/contracts/src/libraries/OptionMath.sol#L70-73>

```
(amount * bps) / 100
```

So the sum of separately rounded collateral amounts can be lower than the collateral required for the combined option supply. For example, for a strike-50 call, two separate mints of three raw option units each require only one collateral unit per mint:

```
Mint 1 collateral: floor(3 * 50 / 100) = 1
```

```
Mint 2 collateral: floor(3 * 50 / 100) = 1
```

```
Total collateral added: 2
```

```
Liability added at maximum payout:
```

```
floor(6 * 50 / 100) = 3
```

The series is therefore undercollateralized by one raw unit. When settlement calculates the holder pool from aggregate supply, `holderPool` exceeds `totalCollateral`, causing the settlement transaction to revert:

<https://github.com/sherlock-audit/2026-06-convallax-june-19th-2026/blob/main/Convallax/contracts/src/ConvallaxCore.sol#L432-434>

```
if (holderPool > s.totalCollateral) {  
    revert InvariantViolation("holderPool > totalCollateral");  
}
```

Impact

An attacker can permanently prevent a series from settling at a valid endpoint price using very small mint amounts and negligible collateral.

Tool Used

Manual Review

Recommendation

Round mint collateral upward so every mint fully covers the liability it creates:

```
+uint256 collateral = Math.mulDiv(  
+    amount,  
+    bps,  
+    100,  
+    Math.Rounding.Ceil  
+);
```

Issue L-1: Zero address cannot disable atomic fills as specified [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-06-convallax-june-19th-2026/issues/10>

Summary

The specification states that setting settlement to `address(0)` disables atomic RFQ fills, but `setSettlement()` rejects the zero address.

Vulnerability Detail

The documented emergency-disable mechanism cannot be used because the function reverts whenever `_settlement` is zero:

<https://github.com/sherlock-audit/2026-06-convallax-june-19th-2026/blob/main/Convallax/audit/SPEC.md#23b-atomic-write-on-fill-mintfor-settlement-only>

```
(settlement is set by the owner via setSettlement; address(0) disables atomic  
↪ fills.)
```

<https://github.com/sherlock-audit/2026-06-convallax-june-19th-2026/blob/main/Convallax/contracts/src/ConvallaxCore.sol#L459-463>

```
function setSettlement(address _settlement) external onlyOwner {  
    if (_settlement == address(0)) revert ZeroAddress();  
    emit SettlementUpdated(settlement, _settlement);  
    settlement = _settlement;  
}
```

Impact

Deviation from the specification. The owner cannot disable `mintFor()` access without replacing the current settlement contract with another nonzero address.

Tool Used

Manual Review

Recommendation

Either permit the zero address or update the specification and provide a separate explicit function for disabling atomic fills.

Issue L-2: Series-ID architecture comment deviates from the code [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-06-convallax-june-19th-2026/issues/11>

Summary

The architecture comment states that `seriesId` is derived from four fields, while the implementation derives it from five fields.

Vulnerability Detail

The comment describes the identifier as:

<https://github.com/sherlock-audit/2026-06-convallax-june-19th-2026/blob/main/Convallax/contracts/src/ConvallaxCore.sol#L21-22>

```
/// - Series registry: each option series is identified by a `seriesId` =  
///   keccak256(underlying, strikeBps, expiry, optionType).
```

However, the actual identifier includes:

<https://github.com/sherlock-audit/2026-06-convallax-june-19th-2026/blob/main/Convallax/contracts/src/ConvallaxCore.sol#L227>

```
return uint256(keccak256(abi.encode(conditionId, yesClobTokenId, strikeBps, expiry,  
↪   optionType))));
```

Impact

Deviation from the specification. Off-chain integrations that reproduce the documented formula may calculate an incorrect `seriesId`.

Tool Used

Manual Review

Recommendation

Update the architecture comment to list all five inputs and include the exact encoding method used by the implementation.

Issue L-3: Documented revert surface does not match OpenZeppelin v5 [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-06-convallax-june-19th-2026/issues/12>

Summary

The specification documents OpenZeppelin v4-style revert strings for ownership, pause, and reentrancy failures, while the OpenZeppelin v5 contracts emit custom errors.

Vulnerability Detail

The specification expects the following string-based reverts:

<https://github.com/sherlock-audit/2026-06-convallax-june-19th-2026/blob/main/Convallax/audit/SPEC.md#6-expectedpotential-errors-and-revert-paths>

```
"Ownable: caller is not the owner"  
"Pausable: paused"  
"ReentrancyGuard: reentrant call"
```

However, the implementation inherits OpenZeppelin v5, which instead emits:

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v5.6.1/contracts/access/Ownable.sol#L23-L26> <https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v5.6.1/contracts/utils/Pausable.sol#L74-L77>

```
OwnableUnauthorizedAccount(address account)  
EnforcedPause()  
ExpectedPause()  
ReentrancyGuardReentrantCall()
```

Therefore, the revert selectors and encoded return data exposed by the contracts differ from the specification's documented public revert ABI.

Impact

Frontends, SDKs relying on the documented revert strings may fail to decode or correctly classify these failures.

Tool Used

Manual Review

Recommendation

Update the revert catalogue to list the exact OpenZeppelin v5 custom errors.

Issue L-4: `mintFor` does not enforce distinct writer and holder addresses [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-06-convallax-june-19th-2026/issues/13>

Summary

The specification states that `mintFor` uses distinct writer and holder parties, but the implementation allows both roles to use the same address.

Vulnerability Detail

The protocol's specification states:

```
mintFor(seriesId, amount, writer, holder) is the primary-issuance primitive used by
↳ RFQ fills. It is identical to mint except the writer and holder are distinct
↳ parties and the caller is gated
```

However, `mintFor` function only verifies that both addresses are nonzero. It does not enforce:

```
writer != holder
```

Therefore, the settlement contract can call `mintFor` with the same address as both writer and holder.

Impact

No direct loss of funds occurs, but the documented `mintFor` behavior does not match the enforced contract behavior.

Tool Used

Manual Review

Recommendation

Either enforce distinct addresses or update the specification to clarify that writer and holder are separate economic roles but may resolve to the same address.

Issue L-5: Constructor inputs are not all stored as immutables [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-06-convallax-june-19th-2026/issues/14>

Summary

The specification states that constructor inputs are recorded as immutable in each contract. In practice, several constructor-provided values, including `resolutionSigner` and `owner`, are stored in mutable contract storage.

Vulnerability Detail

The specification states:

```
Constructor inputs are checked against address(0) and recorded as immutable in each
↳ contract.
```

This does not match `ConvallaxCore`:

- `resolutionSigner` is stored in the mutable `resolutionSigner` variable and can be changed through `setResolutionSigner`
- `_owner` is stored by OpenZeppelin `Ownable` and can be changed through `transferOwnership` or cleared through `renounceOwnership`
- the claim also does not fully hold for `ConvallaxOptionToken`, where `_owner` is mutable `Ownable` storage rather than an immutable value.

Impact

No serious impact, only readers may incorrectly conclude that deployment fixes all constructor-provided authority addresses permanently.

Tool Used

Manual Review

Recommendation

Limit the immutability statement to the values actually declared immutable.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.