Predict.fun Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Farouk

qpzm

Assisting Auditors

Alexzoid (Formal Verification)

February 18, 2026

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

Predict.fun is a prediction market trading platform that extends the Gnosis Conditional Token Framework with yield-bearing capabilities. The protocol allows users to deposit collateral tokens to receive conditional tokens representing positions in prediction markets, while the underlying collateral is simultaneously deployed to Venus protocol on BNB Chain to generate yield. This dual-purpose design enables prediction market participants to benefit from yield generation on their locked collateral.

The core mechanism centers on three primary operations: splitting, merging, and redeeming positions. When users split positions, they deposit collateral which is converted to Venus vTokens, and receive ERC1155 conditional tokens representing their prediction market positions. Merging reverses this process, burning conditional tokens and returning the underlying collateral by redeeming from Venus. After a prediction market resolves, users can redeem their winning positions for proportional payouts. Throughout these operations, the protocol tracks deposited amounts separately from yield, allowing a designated yield manager to claim accrued returns without touching user principal.

The architecture employs a role-based access control system with several key permissions. The DEFAULT_ADMIN_ROLE controls transfer whitelisting and can gate split and merge operations. The YIELD_MANAGER_ROLE manages Venus integration, including connecting vTokens, claiming yield, and enabling or disabling the yield mechanism. Transfer controls are implemented through WhitelistedERC1155, which allows the admin to restrict token transfers to approved addresses. The Venus base contract handles all interactions with the external yield protocol, mapping underlying tokens to their corresponding vTokens.

Supporting components include CTHelpers, a library using elliptic curve cryptography to compute condition, collection, and position identifiers, and YieldBearingWrappedCollateral, an ERC20 wrapper used by the NegRiskAdapter for multi-outcome market scenarios. The protocol includes emergency mechanisms allowing the yield manager to disable Venus integration and withdraw funds if needed.

For the system to remain secure, the Venus integration must maintain accurate accounting between deposited principal and actual vToken value, user redemptions must always be possible regardless of Venus utilization, and the yield manager must not be able to claim more than the actual yield generated.

5 Audit Scope

The audit scope was limited to:

```
contracts/ConditionalTokens/CTHelpers.sol
contracts/ConditionalTokens/WhitelistedERC1155.sol
contracts/YieldBearing/Venus.sol
contracts/YieldBearing/YieldBearingConditionalTokens.sol
contracts/YieldBearing/YieldBearingWrappedCollateral.sol
```

6 Executive Summary

Over the course of 5 days, the Cyfrin team conducted an audit on the Predict.fun smart contracts provided by Predict.fun. In this period, a total of 4 issues were found.

Predict.fun extends Gnosis Conditional Tokens with yield-bearing capabilities, deploying prediction market collateral to Venus protocol on BNB Chain to generate returns. The security model depends on correct accounting between deposited principal and vToken value, uninterrupted access to Venus redemptions, and proper role-based access controls for yield management and transfer restrictions.

Our review focused on the Venus yield integration mechanics, position management flows including split, merge, and redeem operations, access control implementations, the transfer whitelist system, and the elliptic curve cryptography used for collection ID computation. The audit identified 5 findings: 1 Medium, 1 Low, and 3 Informational. No critical vulnerabilities were discovered.

The medium severity finding relates to the emergency disable mechanism for Venus integration: during high utilization periods when withdrawals fail, the disable function itself attempts a withdrawal and therefore cannot execute. This creates a circular dependency where the emergency mechanism is unusable during the exact scenario it was designed to address. The low severity item addresses yield calculation edge cases when Venus vToken exchange rates decrease due to external factors such as bad debt, though Venus does not currently implement debt socialization by default.

Additional findings provide informational recommendations: the hash-to-curve algorithm in CTHelpers uses a variable-iteration approach that could benefit from BLS12-381 precompiles now available on BNB Chain, the Venus integration is limited to V3 Core Pool and incompatible with V4 Isolated Pools, and the transfer whitelist implements OR logic allowing transfers when any party is whitelisted. Our analysis of CTHelpers confirmed that despite the mathematical possibility of creating unusual token positions through negative tying or multiple tying to the same condition, the collateral invariant is always preserved and no funds can be extracted beyond what was deposited.

Overall, the codebase demonstrates a solid security foundation with well-structured access controls and clear separation between conditional token logic and yield generation.

Summary

Project Name	Predict.fun
Repository	prediction-market
Commit	8247e9a6f98e...
Audit Timeline	Jan 21st - Jan 27th, 2026
Methods	Manual Review, Formal Verification

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	1
Informational	2
Gas Optimizations	0
Total Issues	4

Summary of Findings

[M-1] `Venus._disableUnderlying` fails during high utilization	Acknowledged
[L-1] VToken value decrease causes underflow in yield calculations	Acknowledged
[I-1] Transfer Whitelist Bypass via OR Logic in WhitelistedERC1155	Acknowledged
[I-2] Consider upgrading to BLS12-381 constant-time hash-to-curve	Acknowledged

7 Findings

7.1 Medium Risk

7.1.1 `Venus._disableUnderlying` fails during high utilization

Description: The Sherlock M-1 fix (`ef0712b`) added `_disableUnderlying` as an emergency mechanism to disable Venus integration when high utilization blocks withdrawals. However, the function itself calls `redeemUnderlying` to withdraw all funds before disabling:

```
// Venus.sol:259-263
if (redeemAmount > 0) {
    uint256 err = IVToken(vToken).redeemUnderlying(redeemAmount);
    if (err != 0) {
        revert VTokenCallFailed(err);  // ← Reverts during high utilization
    }
}
```

When Venus pool has high utilization, `redeemUnderlying` fails with error code 13 (`TOKEN_INSUFFICIENT_CASH`), causing the entire function to revert.

The circular problem:

```
[Original Problem] High utilization -> redeemUnderlying fails -> merge/redeem broken
[The Fix]          Add _disableUnderlying as emergency mechanism
[Actual Behavior]  High utilization -> _disableUnderlying calls redeemUnderlying -> fails -> cannot
↪  disable
```

Impact: The emergency disable mechanism is unusable during the exact emergency it was designed to handle. All withdrawal operations are blocked with no working admin escape mechanism. A `softDisableUnderlying` function would help by:

1. Immediate flag flip: Sets `underlyingIsEnabled = false` without withdrawal attempt
2. New splits accumulate underlying: With flag disabled, `splitPosition` keeps underlying in contract instead of depositing to Venus
3. Merges can resume: When flag is `false`, merge transfers directly from contract's underlying balance (the else branch)
4. Funds in Venus remain safe: Existing vToken position stays until utilization drops, then admin can withdraw via `disableUnderlying`

Proof of Concept: File: `test/foundry/poc/M-1_DisableUnderlyingHighUtilization.t.sol`

```
forge test --match-contract DisableUnderlyingHighUtilizationPOC -vvv
```

Test	Result
`test_disableUnderlying_failsDuringHighUtilization`	PASS (confirms revert)
`test_fullScenario_emergencyDisableFails`	PASS

Recommended Mitigation: Add a `softDisableUnderlying` function that only flips the flag without attempting withdrawal:

```
event UnderlyingTokenSoftDisabled(address indexed underlying);

/**
 * @notice Soft disables Venus without withdrawal. Use during high utilization.
 * @param underlying The underlying token
 */
```

```
function _softDisableUnderlying(address underlying) internal {
    if (!underlyingIsEnabled[underlying]) {
        revert UnderlyingTokenAlreadyDisabled();
    }
    underlyingIsEnabled[underlying] = false;
    // No withdrawal - funds remain in Venus until utilization drops
    emit UnderlyingTokenSoftDisabled(underlying);
}
```

After soft disable, the flag is already `false`, so `disableUnderlying` would revert. When utilization drops, admin can either call `enableUnderlying` to resume Venus integration, or add a `withdrawRemainingVTokens` function to withdraw remaining vTokens. This creates a two-phase approach:

1. High utilization crisis: call `softDisableUnderlying`
2. Utilization drops: call `enableUnderlying` or `withdrawRemainingVTokens`

Partial Liquidity Safety Analysis:

The `softDisableUnderlying` approach maintains correct accounting. The solvency invariant holds at every step:

```
totalAssets >= totalYieldBearingConditionalTokenSupply

where:
  totalAssets = underlying.balanceOf(contract) + vToken.balanceOfUnderlying(contract)
  totalYieldBearingConditionalTokenSupply = sum of all YieldBearingConditionalToken balances for this
  ↪  underlying
```

Step	Action	Underlying	vToken	YBCT Supply	Invariant	
1	Before crisis (flag=true)	0	1000	1000	1000	1000
2	`softDisableUnderlying` (flag=false)	0	1000	1000	1000	1000
3	New user splits 200 (stays in contract)	200	1000	1200	1200	1200
4	Any user merges 200 (from underlying)	0	1000	1000	1000	1000
5	Utilization drops, full disable	1000	0	1000	1000	1000

Funds are fungible - the protocol doesn't need to track which user deposited when. The only difference between "all-or-nothing" and "partial liquidity" is:

Approach	Liquid funds during crisis	Withdrawals possible
Current (all-or-nothing)	0	0
With `softDisableUnderlying`	New deposits	Yes (first-come-first-served)

Partial liquidity is better: *some* users can withdraw rather than *no* users.

Predict.fun: Acknowledged; as the change in the design is quite huge and probably not justifiable for a redeploy.

7.2 Low Risk

7.2.1 VToken value decrease causes underflow in yield calculations

Description: The `Venus` contract tracks deposited stablecoin amounts in `depositedAmount[underlying]`, but the yield calculations in `Venus::splitPrincipalAndYield` assume the VToken exchange rate only increases. When VToken value decreases (due to bad debt, exploits, or other losses in Venus protocol), the calculations underflow.

```
// Venus.sol:118-123
function splitPrincipalAndYield(address underlying) public returns (uint256 principal, uint256 yield) {
    address vToken = _getVToken(underlying);
    uint256 exchangeRateCurrent = IVToken(vToken).exchangeRateCurrent();
    (, principal) = divScalarByExpTruncate(depositedAmount[underlying], Exp({mantissa:
    ↪  exchangeRateCurrent}));
    yield = IVToken(vToken).balanceOf(address(this)) - principal;  //  Underflows when exchange rate
    ↪  drops
}
```

The `principal` calculation converts the deposited underlying amount to vToken units:

```
principal = depositedAmount / exchangeRate
```

When the exchange rate drops, `principal` increases (more vTokens needed to represent the same underlying). If `principal > vToken.balanceOf(this)`, the yield calculation underflows.

Example scenario:

1. User deposits 1000 USDC at exchange rate 1.0e18 → receives ~1000 vTokens
2. `depositedAmount = 1000e18`
3. Venus experiences bad debt, exchange rate drops to 0.8e18
4. `principal = 1000e18 / 0.8e18 = 1250` vTokens
5. Actual vToken balance is ~1000
6. `yield = 1000 - 1250` → **UNDERFLOW**

A similar issue affects `_disableUnderlying`:

```
// Venus.sol:253-264
uint256 redeemAmount = IVToken(vToken).balanceOfUnderlying(address(this));  // e.g., 800e18 (after bad
↪  debt)
uint256 originalDepositedAmount = depositedAmount[underlying];              // e.g., 1000e18
if (redeemAmount < originalDepositedAmount) {  // 800 < 1000 → true
    redeemAmount = originalDepositedAmount;    // redeemAmount = 1000e18, but only 800e18 available!
}

if (redeemAmount > 0) {
    uint256 err = IVToken(vToken).redeemUnderlying(redeemAmount);  // redeemUnderlying(1000e18) fails
    if (err != 0) {
        revert VTokenCallFailed(err);
    }
}
```

When `balanceOfUnderlying < originalDepositedAmount` (e.g., 800 < 1000 due to bad debt), the code forces `redeemAmount = 1000e18`, but Venus cannot fulfill this because the vTokens are only worth 800e18. Venus returns an error code, causing `VTokenCallFailed` revert.

Impact: When Venus protocol experiences any loss event (bad debt, exploit, interest rate miscalculation):

Function	Effect
`splitPrincipalAndYield`	Reverts due to underflow
`_claimYield`	Reverts (calls `splitPrincipalAndYield`)
`_disableUnderlying`	Reverts with `VTokenCallFailed` (tries to redeem more than available)

The protocol becomes stuck - admin cannot disable Venus integration, and yield operations revert. This creates a situation where:

- The underlying value has already decreased
- No admin action can recover from this state
- All yield-related operations are blocked indefinitely

Proof of Concept: File: `test/foundry/poc/M-2_VTokenValueDecrease.t.sol`

```
forge test --match-contract VTokenValueDecreasePOC -vvv
```

Test	Result
`test_disableUnderlying_revertsWhenExchangeRateDrops`	PASS (`VTokenCallFailed(9)`)
`test_splitPrincipalAndYield_underflowsWhenExchangeRateDrops`	PASS (arithmetic underflow)
`test_claimYield_revertsWhenExchangeRateDrops`	PASS (calls `splitPrincipalAndYield`)

Key scenario from the POC:

```
function test_disableUnderlying_revertsWhenExchangeRateDrops() public {
    // Simulate Venus bad debt: exchange rate drops to 0.8e18 (20% loss)
    vToken.setExchangeRate(0.8e18);

    // Now:
    // - vToken balance: 1000e18 vTokens
    // - balanceOfUnderlying: 1000e18 * 0.8e18 / 1e18 = 800e18 (worth only 800 now)
    // - depositedAmount: 1000e18 (original deposit)
    //
    // _disableUnderlying logic:
    // 1. redeemAmount = balanceOfUnderlying = 800e18
    // 2. originalDepositedAmount = 1000e18
    // 3. if (800 < 1000) -> redeemAmount = 1000e18  <-- tries to redeem more!
    // 4. redeemUnderlying(1000e18) fails because only 800e18 worth available

    vm.expectRevert(abi.encodeWithSelector(Venus.VTokenCallFailed.selector, 9));
    venus.disableUnderlying(address(underlying), yieldRecipient);
}
```

Recommended Mitigation: Minimal fix - Prevent reverts when Venus loses value:

1. In `splitPrincipalAndYield`: Return `(vTokenBalance, 0)` when `vTokensForPrincipal >= vTokenBalance`
2. In `_disableUnderlying`: Redeem only what's available, don't force `originalDepositedAmount`

Note: This minimal fix still allows first-come-first-served withdrawals during a loss scenario. For fair loss distribution, user redemption functions(`mergePositions`, `redeemPositions`) should check health ratio:

- `healthRatio >= 1`: Redeem full principal (no loss)

- `healthRatio < 1`: Redeem `amount * healthRatio` (proportional loss)

Predict.fun: We acknowledge the finding. After investigating Venus protocol's exchange rate mechanics:

Current Venus Behavior:

- Exchange rate formula: `(totalCash + totalBorrows - totalReserves) / totalSupply`
- Venus does NOT implement socialized debt by default
- Liquidations via `liquidateBorrowFresh` require full debt repayment - no bad debt write-off mechanism
- In normal operations, the exchange rate should never decrease

When Can Exchange Rate Decrease?

Scenario	Likelihood	Venus Default?
Normal withdrawals/repayments	N/A	Rate unchanged
Socialized debt	Very Low	**No** - not implemented
Venus exploit	Low	External risk

Cyfrin: Venus vTokens are **upgradeable proxy contracts**:

- Proxy: `0x95c78222B3D6e262426483D42CfA53685A67Ab9D`
- Implementation: `0x33d17f1e6107cd4d711b56eb0094bf39a471a8b5`

Venus governance could upgrade to add socialized debt or modify the exchange rate formula in the future. While this is outside our control, the recommended mitigation provides defense-in-depth against:

1. Venus exploit scenarios
2. Potential future Venus upgrades that introduce debt socialization
3. Any unforeseen mechanism that could decrease the exchange rate

We suggest a fix to handle edge cases gracefully.

7.3 Informational

7.3.1 Transfer Whitelist Bypass via OR Logic in WhitelistedERC1155

Description: The `_isTransferAllowed()` function implements OR logic instead of AND logic using nested IF statements. A transfer is allowed if ANY of `msg.sender`, `from`, or `to` is whitelisted, rather than requiring all parties to be whitelisted.

```
function _isTransferAllowed(address from, address to) internal view {
    if (transferControlEnabled) {
        if (!isTransferAllowed[msg.sender]) {
            if (!isTransferAllowed[from]) {
                if (!isTransferAllowed[to]) {
                    revert NotAllowedToTransfer();  // Only reverts if ALL are false
                }
            }
        }
    }
}
```

Impact: This makes the whitelist protection weaker than expected if the intent is to restrict transfers between untrusted parties.

Recommended Mitigation: If the intent is to require ALL parties to be whitelisted:

```
function _isTransferAllowed(address from, address to) internal view {
    if (transferControlEnabled) {
        if (!isTransferAllowed[msg.sender] || !isTransferAllowed[from] || !isTransferAllowed[to]) {
            revert NotAllowedToTransfer();
        }
    }
}
```

Similar whitelist pattern in Ethena's StakedUSDe uses explicit OR with clear intent:

StakedUSDe.sol#L231-L236

```
function _beforeTokenTransfer(address from, address to, uint256) internal virtual override {
    if (hasRole(FULL_RESTRICTED_STAKER_ROLE, from) || hasRole(FULL_RESTRICTED_STAKER_ROLE, to)) {
        revert OperationNotAllowed();
    }
    // ...
}
```

Predict.fun: Acknowledged, this is intended behavior to allow transfers as long as the msg.sender, from or to is whitelisted.

7.3.2 Consider upgrading to BLS12-381 constant-time hash-to-curve

Description: `CTHelpers::getCollectionId` uses a try-and-increment algorithm inherited from the original Gnosis Conditional Tokens (2019):

```
function getCollectionId(
    bytes32 parentCollectionId,
    bytes32 conditionId,
    uint indexSet
) internal view returns (bytes32) {
    uint x1 = uint(keccak256(abi.encodePacked(conditionId, indexSet)));
    bool odd = x1 >> 255 != 0;
    uint y1;
    uint yy;
    do {
        x1 = addmod(x1, 1, P);
```

```
        yy = addmod(mulmod(x1, mulmod(x1, x1, P), P), B, P);
        y1 = sqrt(yy);
    } while (mulmod(y1, y1, P) != yy); // @audit Variable iterations
    // ...
}
```

This approach has limitations compared to modern alternatives now available via the Pascal upgrade (BNB Chain, March 2025).

Comparison:

Aspect	Current (BN254 Try-and-Increment)	BLS12-381 (MAP_FP_TO_G1 Precompile)
Algorithm	Increment x until x^3+3 is quadratic residue	Simplified SWU (constant-time mapping)
Time complexity	Variable (~50% success per iteration)	Constant (single precompile call)
Gas cost	~2,000-5,000 typical, can be higher	~5,500 (fixed)
Security level	~100 bits	~128 bits
Timing side-channel	Vulnerable (iteration count leaks info)	Resistant (constant-time)
Precompile	`ecAdd` (0x06), `ecMul` (0x07)	`MAP_FP_TO_G1` (0x12)
Output size	32 bytes (compressed)	48 bytes (compressed)
Availability	All EVM chains	Pectra+ chains (Ethereum May 2025, BNB Pascal March

Impact: This is not a vulnerability in the current implementation. Our loop analysis shows the worst case is bounded (max 26 iterations in 16M+ samples, ~85k gas). However, the try-and-increment method has:

1. **Variable gas costs:** Worst-case iterations can exceed typical gas estimates (~20k average vs ~85k worst case)
2. **Timing side-channel:** Iteration count reveals information about the input hash
3. **Lower security margin:** BN254 provides ~100 bits vs BLS12-381's ~128 bits

Recommended Mitigation: For future upgrades, consider migrating to BLS12-381 with the `MAP_FP_TO_G1` precompile (address 0x10) introduced in EIP-2537. This provides:

- Constant-time execution (no timing side-channel)
- Fixed gas cost (predictable)
- Higher security level (128 bits)

```
// BLS12-381 precompile addresses (EIP-2537)
address constant BLS12_G1ADD = address(0x0b);         // 375 gas
address constant BLS12_MAP_FP_TO_G1 = address(0x10); // 5500 gas (constant-time SWU)

function hashToG1(bytes32 conditionId, uint256 indexSet) internal view returns (bytes memory) {
    // 1. Hash to 64-byte field element
    bytes memory fieldElement = new bytes(64);
    bytes32 hash = keccak256(abi.encodePacked(conditionId, indexSet));
    assembly {
        mstore(add(fieldElement, 0x40), hash) // Store hash in lower 32 bytes
    }

    // 2. Single precompile call - constant time, no iterations
    (bool success, bytes memory point) = BLS12_MAP_FP_TO_G1.staticcall(fieldElement);
    require(success, "MAP_FP_TO_G1 failed");

    return point; // 128 bytes (uncompressed G1 point)
```

```
}
```

Note: Migration would require updating `collectionId` format from 32 bytes to 48 bytes (compressed G1 point), affecting storage and existing integrations.

References:

- EIP-2537: BLS12-381 curve operations
- Gnosis Conditional Tokens (2019)
- hash_to_curve_comparison.md
- CTHelpers-getCollectionId-loop-analysis.md - Worst-case loop iteration analysis

Predict.fun: Acknowledged, will keep it as it is since the code is originally from Gnosis Conditional tokens. Making a change would change the security guarantees to be the same as upstream code.