SHERLOCK

Security Review For Predict.fun

Collaborative Audit Prepared For: **Predict.fun**
Lead Security Expert(s): **jokr**
oot2k
Date Audited: **October 27 - November 10, 2025**

Introduction

Introducing predict.fun - the prediction market with DeFi capital efficiency

Scope

Repository: PredictDotFun/prediction-market

Audited Commit: 8eafbc15898878911c8e7073d1f65dc5a7e7437d

Final Commit: 5aaf72d381e8580740b32dbf66a8bcf1a5424e2e

Files:

- contracts/ConditionalTokens/ConditionalTokens.sol
- contracts/ConditionalTokens/CTHelpers.sol
- contracts/ConditionalTokens/WhitelistedERC1155.sol
- contracts/NegRisk/libraries/CTHelpers.sol
- contracts/NegRisk/libraries/Helpers.sol
- contracts/NegRisk/libraries/NegRiskIdLib.sol
- contracts/NegRisk/modules/MarketDataManager.sol
- contracts/NegRisk/types/MarketData.sol
- contracts/NegRisk/YieldBearingNegRiskAdapter.sol
- contracts/OptimisticOracle/common/implementation/Lockable.sol
- contracts/OptimisticOracle/data-verification-mechanism/implementation/Constants.sol
- contracts/OptimisticOracle/interfaces/OptimisticOracleV2Interface.sol
- contracts/OptimisticOracle/UmaCompatibleOptimisticOracle.sol
- contracts/UMA/interfaces/IBulletinBoard.sol
- contracts/UMA/interfaces/IOptimisticOracleV2.sol
- contracts/UMA/interfaces/IUmaCtfAdapter.sol
- contracts/UMA/libraries/AncillaryDataLib.sol
- contracts/UMA/libraries/PayoutHelperLib.sol
- contracts/UMA/UmaCompatibleCtfAdapter.sol
- contracts/YieldBearing/Venus.sol
- contracts/YieldBearing/YieldBearingConditionalTokens.sol
- contracts/YieldBearing/YieldBearingWrappedCollateral.sol

Final Commit Hash

5aaf72d381e8580740b32dbf66a8bcf1a5424e2e

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
0	4	3

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue M-1: High utilization of Venus pool will break all withdraw operations [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/issues/23

Summary

Venus vTokens allow to deposit funds which can be lend or used as collateral. When the utilization reaches 100%, every token that is currently in the pool is borrowed, meaning no withdraws are possible anymore. This can happen for various reasons, for example in case large withdraws are made or collateral is liquidated.

As prediction markets depend on deposits and withdraws, any issue with token withdraws will break the core logic of the market.

Vulnerability Detail

Predict.fun `Venus.sol` redeems underlying tokens from Venus.

```
function _redeemUnderlying(address underlying, uint256 amount) internal {
    address vToken = _getVToken(underlying);

    uint256 err = IVToken(vToken).redeemUnderlying(amount);
```

`_redeemUnderlying` is called in `YieldBearingWrappedCollateral.sol` and `YieldBearingConditionalTokens.sol`.

Impact

The functions `Merge`, `redeem` as well as `convert` for negrisk will break. In the worst case, the pool has accrued bad debt and user deposits will diminish.

Code Snippet

https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/YieldBearing/YieldBearingWrappedCollateral.sol#L104
https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/YieldBearing/YieldBearingWrappedCollateral.sol#L141
https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/YieldBearing/YieldBearingConditionalTokens.sol#L268
https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/YieldBearing/YieldBearingConditionalTokens.sol#L329

Tool Used

Manual Review

Recommendation

Even if high utilization is unlikely for pools with large TVL smaller pools might experience these issues. We recommend to not integrate with them. Additionally a kill switch should be implemented to stop market creation as well as deposits in case utilization is high.

Discussion

0xnostredame

https://github.com/PredictDotFun/prediction-market/pull/28

Issue M-2: In case supply caps are reached in Venus, markets wont accept deposits [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/issues/24

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Venus defines supply caps on each deposited asset. This is to prevent over exposure to a certain token. Predict.fun deposits tokens into Venus to earn yield. In case the supply caps are reached, deposits are blocked and markets that use this token will break.

Vulnerability Detail

Following mint function of `Venus.sol`, which is used in `YieldBearingWrappedCollateral.sol` and `YieldBearingConditionalTokens.sol`, will revert.

```
function _mintVToken(address underlying, uint256 amount) internal {
    address vToken = _getVToken(underlying);

    depositedAmount[underlying] += amount;

    IERC20(underlying).forceApprove(vToken, amount);
    uint256 err = IVToken(vToken).mint(amount);
```

This will break the `split` function of conditional tokens, not allowing new capital to enter the market.

Impact

This will break the `split` function of conditional tokens.

Code Snippet

https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/YieldBearing/Venus.sol#L81

Tool Used

Manual Review

Recommendation

Fixing this on the smart contract level is rather complex / would require large code changes. We recommend to use pools with high supply caps, and monitor market creation.
Because only deposits are blocked the risk to user funds is rather low.

Discussion

0xnostredame

Ack

Issue M-3: Incorrect check allows to claim full yield [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/issues/27

Summary

Predict.fun implements a yield claiming mechanism to allow a trusted operator to claim yields generated on Venus. The protocol defines that the claimed yield should always be smaller then the actual claimable amount, due to rounding errors. Currently this is not correctly enforced.

Vulnerability Detail

The `_claimYield` of `Venus.sol` does following "yield adjustment" to enforce the full yield is never claimed.

```
if (vTokenAmount == 0) {
    vTokenAmount = (yield * 9_999) / 10_000;
} else if (vTokenAmount > yield) {
    revert CannotWithdrawMoreThanYield();
}
```

This check however fails to consider the case in which vTokenAmount = yield. If this accrues the yield is not adjusted, breaking core invariant.

Impact

Core protocol assumption of "We should never claim 100% of the yield because Venus' balanceOfUnderlying carries precision loss" is broken.

Code Snippet

https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/YieldBearing/Venus.sol#L123-L127

Tool Used

Manual Review

Recommendation

We recommend checking adjusting the total claim amount in case vTokenAmount == yield.

Discussion

0xnostredame

https://github.com/PredictDotFun/prediction-market/pull/26

0xnostredame

Merged the fix to the wrong branch, this is the updated PR
https://github.com/PredictDotFun/prediction-market/pull/34

Issue M-4: In case Venus introduces a protocol fee merge as well as redeem function of conditional tokens will break [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/issues/29

Summary

In case Venus introduces a protocol fee, it will be taken on `redeem`. This means the redeeming user receives less tokens then what he requested.

Because Predict.fun assumes that Venus redeem always returns the same amount of tokens as requested, it tries to transfer this amount to the user. In case Venus activates fees, this transaction will revert.

Vulnerability Detail

Following function redeems the exact amount of provided tokens from Venus.

```
function _redeemUnderlying(address underlying, uint256 amount) internal {
        address vToken = _getVToken(underlying);

        uint256 err = IVToken(vToken).redeemUnderlying(amount);
        if (err != 0) {
            revert VTokenCallFailed(err);
        }

        depositedAmount[underlying] -= amount;

        emit UnderlyingTokenRedeemed(underlying, vToken, amount);
    }
```

https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/YieldBearing/Venus.sol#L103

This is used in for example `YieldBearingWrappedCollateral`, where the contract assumes the returned amount of underlying token is the same as requested.

```
function unwrap(address _to, uint256 _amount) external {
    _burn(msg.sender, _amount);
    _redeemUnderlying(underlying, _amount);
    ERC20(underlying).safeTransfer(_to, _amount);
}
```

https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/YieldBearing/YieldBearingWrappedCollateral.sol#L102C1-L106C6

Similar assumption is taken in `YieldBearingConditionalTokens`:
https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/YieldBearing/YieldBearingConditionalTokens.sol#L268
https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/YieldBearing/YieldBearingConditionalTokens.sol#L329

One of the main definitions in a prediction market is that a pair of outcome tokens always add up to one collateral. In case a fee is introduced, this assumption is broken.

Impact

In case Venus introduces a protocol fee, all markets that are yet to resolve will break.

Code Snippet

https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/YieldBearing/YieldBearingConditionalTokens.sol#L268
https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/YieldBearing/YieldBearingConditionalTokens.sol#L329
https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/YieldBearing/YieldBearingWrappedCollateral.sol#L102C1-L106C6

Tool Used

Manual Review

Recommendation

Because this breaks a core definition of the conditional token framework, we see the only realistic fix to use yield/external capital to cover the fee in case this should ever accrue.

Discussion

0xnostredame

https://github.com/PredictDotFun/prediction-market/pull/29

Issue L-1: Delay Period in NegRiskOperator.sol is unnecessary [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/issues/25

Summary

Predict.fun intends to use a trusted price provider for the oracle. This makes a delay period in the `NegRiskOperator` possibly unnecessary. In case of fast resolving markets, it might be to strict to check for a delay, if the same entity can propose a price and resolve the market.

Vulnerability Detail

The `NegRiskOperator`, which is responsible to operate the `YieldBearingNegRiskAdapter.sol` includes a default delay of 1 hour.

```
function resolveQuestion(bytes32 _questionId) external onlyNotFlagged(_questionId) {
    uint256 reportedAt_ = reportedAt[_questionId];

    if (reportedAt_ == 0) revert ResultNotAvailable();
    if (block.timestamp < reportedAt_ + DELAY_PERIOD) {
        revert DelayPeriodNotOver();
    }
```

This can be seen as unnecessary in current code context.

Impact

A mandatory 1 hour delay for market resolution, even if all entities are trusted.

Code Snippet

https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/NegRisk/NegRiskOperator.sol#L182

Tool Used

Manual Review

Recommendation

Remove delay from the `NegRiskOperator`. ONLY if its indented to be used with the new trusted setup! This can also be left as is for extra security, based on design.

Discussion

0xnostredame

https://github.com/PredictDotFun/prediction-market/pull/22

Issue L-2: Remove dead code in `UmaCompatibleOptimisticOracle.sol` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/issues/26

Summary

There are some unreachable code parts in `UmaCompatibleOptimisticOracle.sol`. These can be removed without compromising interface compatibility.

Vulnerability Detail

`_settle` function includes following check:

```
if (state == State.Expired) {
    // In the expiry case, just pay back the proposer's bond and final fee along
    ↪  with the reward.
    request.resolvedPrice = request.proposedPrice;
} else if (state == State.Resolved) {
    revert("_settle: not settleable");
} else {
    revert("_settle: not settleable");
}
```

This can be simplified to a single else. Theoretically settlement can be made instant as there is no dispute mechanism in a fully trusted setup.

```
// Temporarily disable the re-entrancy guard early to allow the caller to take an
↪  OO-related action inside this callback.
_startReentrantGuardDisabled();
// Callback.
if (requester.code.length > 0 && request.requestSettings.callbackOnPriceSettled) {
    OptimisticRequester(requester).priceSettled(identifier, timestamp,
    ↪  ancillaryData, request.resolvedPrice);
}
_endReentrantGuardDisabled();
```

The callback part of the `_settle` can never be reached as we do not support callbacks. The code can theoretically be removed.

Code Snippet

https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/OptimisticOracle/UmaCompatibleOptimisticOracle.sol#L481-L488

https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/OptimisticOracle/UmaCompatibleOptimisticOracle.sol#L501-L507

Recommendation

Remove unused code

Discussion

0xnostredame

https://github.com/PredictDotFun/prediction-market/pull/23

Issue L-3: [Self Report] Unify access control [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/issues/28

Summary

Predict.fun intends to protect merge, split and convert by access control. This is currently not implemented and the implementation of the conditional token as well as the neg risk adapter differ.

Vulnerability Detail

Currently only splitter role is implemented:

```
function splitPosition(
    IERC20 collateralToken,
    bytes32 parentCollectionId,
    bytes32 conditionId,
    uint[] calldata partition,
    uint amount
) external {
    if (accessControlEnabled) {
        _checkRole(SPLIT_POSITION_ROLE);
    }
```

https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/ConditionalTokens/ConditionalTokens.sol#L141

And it is different from the NegRiskAdapter implementation:

```
function splitPosition(bytes32 _conditionId, uint256 _amount) public {
    if (accessControlEnabled && admins[msg.sender] != 1) revert NotAdmin();
```

Impact

Merge and Convert are not protected.

Code Snippet

https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/ConditionalTokens/ConditionalTokens.sol#L141
https://github.com/sherlock-audit/2025-10-predict-fun-oct-27th/blob/main/prediction-market/contracts/NegRisk/NegRiskAdapter.sol#L151C1-L153C1

Tool Used

Manual Review

Recommendation

This was partly reported by the protocol team, a single role for each operation should be implemented.

Discussion

0xnostredame

https://github.com/PredictDotFun/prediction-market/pull/31

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.