SMART CONTRACT AUDIT

ZOKYO.

March 23rd, 2022 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

TECHNICAL SUMMARY

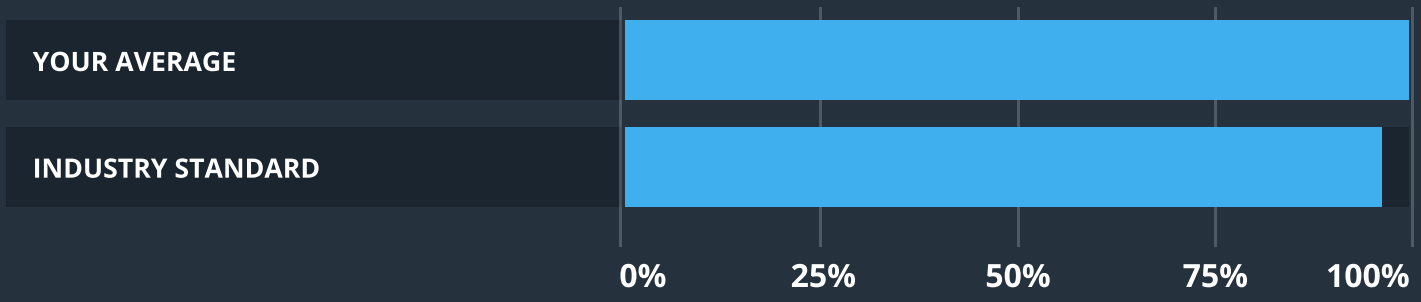
This document outlines the overall security of the Predy smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Predy smart contract codebase for quality, security, and correctness.

Contract Status

There were some no critical, high and medium issues found during the audit.

Testable Code



The testable code is 99%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a security of the contract we at Zokyo recommend that the Predy team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Predy repository.
Repository: https://github.com/predyprotocol/v2-contracts (main branch)
Initial comit:
804e7a81df38f027c4982c1105c4316c05eb6345
Last audited commit:
9c4873bd05296e2b923970d4f6654e9756650dd0

Within the scope of this audit Zokyo auditors have reviewed the following contract(s):

- FeePool.sol
- FlashHedge.sol
- PerpetualMarket.sol
- PerpetualMarketCore.sol
- VaultNFT.sol
- BaseLiquidityPool.sol
- BaseFlashSwap.sol
- EntryPriceMath.sol
- IndexPricer.sol
- Math.sol
- NettingLib.sol
- PoolMath.sol
- SpreadLib.sol
- TraderVaultLib.sol

Throughout the review process, care was taken to ensure that the contract:

- Implements and adheres to existing standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of Predy smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1	Due diligence in assessing the overall code quality of the codebase.	3	Testing contract logic against common and uncommon attack vectors.
2	Cross-comparison with other, similar smart contracts by industry leaders.	4	Thorough, manual review of the codebase, line-by-line.

EXECUTIVE SUMMARY

Auditors team has verified the whole pack of the contracts. Contracts have very good structure and high quality documentation. Also, Predy team has prepared high quality original unit test coverage, which was evaluated as sufficient for the security of the contracts.

There were no critical issues found during the audit. All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner.

Contracts are well written and structured. The findings during the audit have no impact on contract performance or security, so it is fully production-ready.

Nevertheless, all mentioned issues were successfully resolved by the Predy team.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

MEDIUM | RESOLVED

Use SafeERC20

BaseLiquidityPool.sol
FlashHedge.sol
PerpetualMarket.sol
ERC20 tokens should be transferred with 'safeTransfer' and 'safeTransferFrom'. SafeERC20 performs all the checks that tokens were transferred, including ono-standard ERC20 implementation(like in USDT tokens).

Recommendation:
Use 'safeTransfer' and 'safeTransferFrom' instead.

MEDIUM | RESOLVED

No checks for overflow/underflow.

Math.sol, functions addDelta(), scale(), logTaylor(), logTaylor1()
Arithmetical operations should be checked in order not to overflow or underflow.

Recommendation:
Use SafeMath library or update Solidity version to 0.8.x, since starting with this version, Solidity has built-in asserts for overflow/underflow.

MEDIUM | RESOLVED

Unsafe type cast.

PerpetualMarket.sol, function deposit(), line 110.
Such unsafe cast can also lead to incorrect shares calculation,, since user can pass a value greater than uint128,
shares would be minted with this value, but the actual transferred amount would be lower.

Recommendation:
Use SafeCast.

INFORMATIONAL | RESOLVED

Check constructor parameters.

FlashHegde.sol: constructor().
All address parameters should be verified in order not to be zero address.

Recommendation:
Add 'require' statements to check that address parameters are not zero addresses.

	PerpetualMarket.sol	FeePool.sol	FlashHedge.sol
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions / Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	PerpetualMarketCore.sol	VaultNFT.sol	BaseFlashSwap.sol
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions / Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	BaseLiquidityPool.sol	EntryPriceMath.sol	NettingLib.sol
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions / Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	IndexPricer.sol	Math.sol	TraderVaultLib.sol
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions / Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	PoolMath.sol	SpreadLib.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by the Predy team

As part of our work assisting Predy team in verifying the correctness of their contract code, our team has checked the complete set of unit tests prepared by the Predy team.

It needs to be mentioned, that the original code has a significant original coverage with testing scenarios provided by the Predy team. All of them were also carefully checked by the auditors' team to be consistent and meaningful.

```
Contract: EntryPriceMath
  updateEntryPrice
    no positions
      ✓ add long position
      ✓ add short position
  tradePrice not changed
      there are long positions
      ✓ add long position
      ✓ add short position
      ✓ close all positions
      ✓ add short position and positions becomes negative
  there are short positions
      ✓ add short position
      ✓ add long position
      ✓ close all positions
      ✓ add long position and positions becomes positive
  tradePrice becomes high
      there are long positions
      ✓ add long position
      ✓ add short position
      ✓ close all positions
      ✓ add short position and positions becomes negative
  there are short positions
      ✓ add short position
      ✓ add long position
      ✓ close all positions
      ✓ add long position and positions becomes positive
  tradePrice becomes low
      there are long positions
      ✓ add long position
      ✓ add short position
```

✓ close all positions
✓ add short position and positions becomes negatve

there are short positions

✓ add short position
✓ add long position
✓ close all positions
✓ add long position and positions becomes positive

tradePrice and entryPrice are same and values are negative

there are long positions

✓ add long position
✓ add short position
✓ close all positions
✓ add short position and positions becomes negative

there are short positions

✓ add short position
✓ add long position
✓ close all positions
✓ add long position and positions becomes positive

tradePrice is positive and entryPrice is negative

there are long positions

✓ add long position
✓ add short position
✓ close all positions
✓ add short position and positions becomes negative

there are short positions

✓ add short position
✓ add long position
✓ close all positions
✓ add long position and positions becomes positive

tradePrice is negative and entryPrice is positive

there are long positions

✓ add long position
✓ add short position
✓ close all positions
✓ add short position and positions becomes negative

there are short positions

✓ add short position
✓ add long position
✓ close all positions
✓ add long position and positions becomes positive

Contract: **FeePool**

withdraw

✓ withdraw profit (68ms)
✓ reverts if caller is not owner (39ms)

Contract: **FlashHedge**

constructor

✓ reverts if usdc address is zero
✓ reverts if weth address is zero
✓ reverts if Perpetual Market address is zero
✓ reverts if Uniswap Factory address is zero
✓ reverts if ETH-USDC pool address is zero

Contract: **IndexPricer**

calculatePrice

✓ reverts if productId is greater than 2

productId is 1

✓ calculate price when spot is $2,000
✓ calculate price when spot is $3,000
✓ calculate price when spot is $4,000
✓ calculate price when spot is $10,000

productId is 0

✓ calculate price when spot is $1,000

calculateDelta

✓ reverts if productId is greater than 2

productId is 0

✓ calculate delta when spot is $2,000
✓ calculate delta when spot is $3,000
✓ calculate delta when spot is $4,000
✓ calculate delta when spot is $10,000

productId is 1

✓ calculate delta when spot is $1,000

calculateGamma

✓ reverts if productId is greater than 2
✓ gamma is 2
✓ gamma is 0

Contract: **FlashHedge**

hedgeOnUniswap

✓ buy ETH to hedge (414ms)
✓ reverts if net delta is positive (299ms)
✓ reverts if ETH price in Uniswap is too high (390ms)

net delta is negative

✓ sell ETH to hedge (452ms)
✓ reverts if ETH price in Uniswap is too low (330ms)

net delta is negative because the pool has short perpetual future positions

✓ net delta is positive and sell all ETH to hedge (431ms)

Contract: liquidation
  single sub-vaults
    ✓ reverts if the vault has enough margin (268ms)
    ✓ reverts if the vault holds no positions (254ms)
    ✓ liquidate an insolvent vault (327ms)
  withdraw all USDC after the vault liquidated
    ✓ liquidate a vault (580ms)
    ✓ liquidate a vault by funding payment (245ms)
  usdc position is negative
    ✓ liquidate a vault (348ms)
  multiple sub-vaults
    ✓ liquidate a vault (380ms)
    ✓ liquidate a vault by funding payment (378ms)
    ✓ liquidate an insolvent vault (419ms)
    ✓ reverts if the vault has enough margin (230ms)
Contract: Math
  addDelta
    ✓ reverts on overflow because y is too small
    ✓ reverts on overflow because y is too large
  scale
    ✓ scale small number from decimal 6 to 2
    ✓ scale decimal 6 to 2
    ✓ scale decimal 2 to 6
    ✓ scale decimal 6 to 6
    ✓ reverts on overflow
  log
    ✓ returns a correct value for a number of cases (216ms)
    ✓ reverts on overflow
  exp
    ✓ returns a correct value for a number of cases (70ms)
    ✓ reverts on overflow
Contract: NettingLib
  getRequiredMargin
    ✓ no positions
    ✓ short sqeeth
    ✓ short future
    ✓ short sqeeth and long future
  calculateWeightedDelta
    ✓ no positions
    ✓ short sqeeth
    ✓ short future
  addMargin

✓ no positions
✓ short sqeeth
✓ short future
✓ short sqeeth and long future

delta of perpetual future is negative

✓ delta increases
✓ delta increases but no enough usdc
✓ delta becomes 0
✓ delta becomes positive

deltas are negative

✓ delta increases
✓ delta increases but no enough usdc
✓ delta becomes 0
✓ delta becomes positive

delta of squared is negative and delta of future is positive

✓ delta decreases
✓ delta becomes 0
✓ delta becomes negative

getRequiredTokenAmountsForHedge

neutral

✓ deltas are negative
✓ delta of squared perpetual is negative and delta of perpetual future is positive
✓ deltas are positive

underlying positions are positive

✓ deltas are negative
✓ delta of squared perpetual is negative and delta of perpetual future is positive
✓ delta are positive
✓ delta are greater than underlying positions

underlying position of squared perpetual is positive and underlying position of perpetual future is negative

✓ deltas are negative
✓ delta of squared perpetual is negative and delta of perpetual future is positive
✓ deltas are positive

complete

✓ reverts if there are no positions (42ms)

short squared perpetual

✓ spot price not changed
✓ spot price becomes high
✓ reverts if spot price becomes very high
✓ spot price becomes low

short perpetual future

✓ spot price not changed

```
    ✓ spot price becomes high
    ✓ spot price becomes low (44ms)
  short squared perpetual and short perpetual future
    ✓ spot price not changed
    ✓ spot price becomes high
    ✓ spot price becomes low
  short squared perpetual and long perpetual future
    ✓ spot price not changed
    ✓ spot price becomes high
    ✓ spot price becomes low
Contract: hedge
  rebalance required
    ✓ enough USDC locked for a hedge (156ms)
Contract: PerpetualMarket
  initialize
    failure cases
    ✓ reverts if amount is 0
    ✓ reverts if funding rate is 0
  success cases
    ✓ initialize pool
  deposit
    failure cases
    ✓ reverts if amount is 0
  success cases
    ✓ deposit (51ms)
    ✓ withdrawal works after deposit (114ms)
  unrealized PnL > 0 and realized PnL > 0
    ✓ deposit (124ms)
    ✓ withdrawal works after deposit (255ms)
    ✓ large amount of deposit (327ms)
  unrealized PnL < 0 and realized PnL < 0
    ✓ deposit (127ms)
    ✓ withdrawal works after deposit (244ms)
  withdraw
    ✓ reverts if amount is 0
    ✓ reverts if caller is not position owner
    ✓ reverts if withdraw with closing but there are no liquidity (183ms)
    ✓ withdraw all (54ms)
  withdraw all liquidity
    ✓ some trades (200ms)
    ✓ liquidation happened (464ms)
    ✓ hedge (1088ms)
```

tokenPrice becomes high

- ✓ withdraw all (102ms)
- ✓ LP token price is not changed (136ms)

tokenPrice becomes low

- ✓ withdraw all (100ms)
- ✓ LP token price is not changed (144ms)

trade

- ✓ variance updated (669ms)
- ✓ reverts by deadline (201ms)
- ✓ reverts if try to trade with the vault that has no margin (241ms)
- ✓ reverts if try to withdraw from the vault that has no margin (79ms)
- ✓ open multiple vaults (426ms)
- ✓ use multiple sub-vaults (687ms)
- ✓ reverts if trade amount is too large (124ms)
- ✓ reverts if trade amount is too small (59ms)
- ✓ reverts if there is no liquidity (240ms)

limit price

- ✓ reverts long by limit price (268ms)
- ✓ reverts short by limit price (255ms)

access control

- ✓ reverts if caller is not vault owner (190ms)
- ✓ reverts if vault does not exist

Squeeth

- ✓ open position and emit an event (194ms)
- ✓ open short (254ms)
- ✓ close position (357ms)
- ✓ close position with profit (365ms)
- ✓ close position with loss (366ms)

Future

- ✓ open long (223ms)
- ✓ open short (230ms)
- ✓ close (356ms)
- ✓ close with profit (356ms)
- ✓ close with loss (556ms)

Squeeth and Future

- ✓ open Squeeth and Future contracts (332ms)
- ✓ close positions (487ms)
- ✓ close Squeeth (432ms)
- ✓ close Future (435ms)
- ✓ close positions with price move (485ms)
- ✓ large position (544ms)

long squeeth and short future

✓ open long sqeeths and short futures (324ms)
✓ close positions (485ms)
✓ price becomes high and close positions (485ms)
✓ price becomes low and close positions (487ms)

execHedge

✓ net delta is decreased (345ms)
✓ nothing happens if net delta is positive (399ms)

net delta is negative

✓ net delta increased (372ms)
✓ net delta becomes 0 (339ms)

funding payment

✓ pool receives funding fee from squeeth positions (507ms)
✓ pool receives from positive funding fee of future positions (433ms)
✓ pool receives from negative funding fee of future positions (499ms)

getMinCollateralToAddPosition

✓ get min collateral of 0 positions (41ms)
✓ get min collateral of the vault that has no positions (38ms)
✓ get min collateral of squared perpetual and perpetual future (40ms)
✓ get min collateral of the vault that has positions (231ms)

setFeeRecepient

✓ set recepient address
✓ reverts if caller is not owner
✓ reverts if address is 0

Contract: PerpetualMarketCore

initialize

✓ reverts if caller is not PerpetualMarket
✓ suceed to initialize (40ms)

deposit

✓ reverts if caller is not PerpetualMarket
✓ suceed to deposit (44ms)
✓ deposits after that pool position increased (143ms)
✓ deposits after pool gets profit (143ms)
✓ deposits after pool gets loss (143ms)

withdraw

✓ reverts if caller is not PerpetualMarket
✓ withdraws a half of liquidity (47ms)
✓ withdraws all (39ms)
✓ reverts withdrawal if there is little available liquidity (70ms)
✓ withdraws after the pool gets profit (144ms)
✓ withdraws after the pool gets loss (142ms)
✓ spread becomes high (171ms)
✓ spread returns low after time passed (181ms)

✓ spread becomes high when withdraw (168ms)
✓ spread returns low when withdraw (179ms)

updatePoolPosition

✓ reverts if caller is not PerpetualMarket
✓ reverts if pool has no liquidity

after initialized

✓ reverts if pool has no enough liquidity (125ms)
✓ trade price increased as squeeth position increased (252ms)
✓ trade price decreased as squeeth position decreased (433ms)
✓ trade price increased as future position increased (274ms)
✓ trade price decreased as future position decreased (241ms)

locked margin becomes 0 if all positions are closed

✓ close short positions of squared (102ms)
✓ close long positions of squared (99ms)
✓ close short positions of future (98ms)
✓ close long positions of future (97ms)

pool has short position

✓ trade price decreased as position decreased (309ms)
✓ trade price decreased as position size decreased (745ms)

pool has long position

✓ trade price increased as position increased (313ms)
✓ trade price increased as position size increased (749ms)

check utilization

✓ utilization ratio becomes high (55ms)

updatePoolSnapshot

✓ reverts if caller is not PerpetualMarket
✓ variance is not updated (65ms)
✓ variance becomes low (89ms)
✓ variance becomes high (101ms)

getTokenAmountForHedging

✓ get token amounts with min slippage tolerance (85ms)
✓ slippage tolerance becomes big price move (96ms)

delta is negative

✓ get token amounts with min slippage tolerance (119ms)
✓ slippage tolerance becomes big by price move (129ms)

setSquaredPerpFundingMultiplier

✓ set squaredPerpFundingMultiplier
✓ reverts if caller is not owner
✓ reverts if value is negative
✓ reverts if value is greater than 200 * 1e6

setPerpFutureMaxFundingRate

✓ set perpFutureMaxFundingRate

✓ reverts if caller is not owner
✓ reverts if value is negative
✓ reverts if value is greater than 1 * 1e6

setHedgeParams

✓ set hedge params
✓ reverts if caller is not owner
✓ reverts if value is negative
✓ reverts if slippageTolerance is greater than 200
✓ reverts if min is greater than max

setPoolMarginRiskParam

✓ set setPoolMarginRiskParam
✓ reverts if caller is not owner
✓ reverts if value is negative

setTradeFeeRate

✓ set trade fee
✓ reverts if caller is not owner
✓ reverts if value is negative
✓ reverts if trade fee rate is less than protocol fee rate

setPerpetualMarket

✓ reverts if caller is not owner

testUpdateVariance

✓ 12 hours (54ms)
✓ 24 hours (48ms)
✓ 36 hours (38ms)

executeFundingPayment

✓ initialize timestamp
✓ nothing happens if last timestamp is older than now

testGetSignedMarginAmount

✓ short (51ms)
✓ long (52ms)

calculateSignedDeltaMargin

✓ short to short
✓ long to long
✓ short to long
✓ long to short

calculateUnlockedLiquidity

✓ lockedLiquidityAmount=100, deltaM=100, hedgePositionValue=100
✓ lockedLiquidityAmount=100, deltaM=99, hedgePositionValue=99
✓ lockedLiquidityAmount=100, deltaM=101, hedgePositionValue=101
✓ lockedLiquidityAmount=100, deltaM=50, hedgePositionValue=98
✓ lockedLiquidityAmount=100, deltaM=50, hedgePositionValue=102

calculateResultOfFundingPayment

```
    ✓ pool receives funding fee from short perpetual future position
    ✓ pool receives funding fee from long perpetual future position (58ms)
    ✓ pool receives funding fee from squared perpetual position
    ✓ pool pays funding fee from squared perpetual position (57ms)
  calculateFundingRate
    ✓ as the pool position becomes longer, the funding rate becomes smaller (787ms)
    ✓ reverts if liquidity is 0
    ✓ return 0 if product id is invalid
Contract: PoolMath
  calculateMarginDivLiquidity
    ✓ reverts if liquidity is 0
    ✓ return a correct value (1462ms)
Contract: SpreadLib
  getUpdatedPrice
     long
      after safety period
    ✓ maxShortTradePrice is 80
    ✓ maxShortTradePrice is 120
  before safety period
    ✓ maxShortTradePrice is 80
    ✓ maxShortTradePrice is 120
     short
      after safety period
    ✓ minLongTradePrice is 80
    ✓ minLongTradePrice is 120
  before safety period
    ✓ minLongTradePrice is 80
    ✓ minLongTradePrice is 120
  trade
    trade
     payoff check
    ✓ trader and LP gets correct payoff (6826ms)
    ✓ trader and LP gets correct payoff(multiple subVaults) (9312ms)
    ✓ trader and LP gets correct funding received (1974ms)
    ✓ trader and LP gets correct funding received(future's funding rate is positive) (2676ms)
  large amount of trade
    ✓ withdraw all and check balance of PerPetualMarket (117ms)
  getTradePrice
    ✓ get trade price of squared perpetual
    ✓ get trade price of perpetual future
    ✓ reverts if trade amount is too large (88ms)
  pool has short position
```

✓ reverts if trade amount is too small
✓ check trade price (133ms)
pool has long position
✓ reverts if trade amount is too large
✓ check trade price (127ms)
large amount of liquidity
position increased
✓ get squared perpetual's trade price of large position
✓ get perpetual future's trade price of large position
Contract: TraderVaultLib
updateVault
✓ single sub-vault (139ms)
✓ multiple products (180ms)
✓ multiple sub-vaults (88ms)
✓ reverts if sub-vault index is too large
getMinCollateral
single sub-vault
✓ future (556ms)
✓ squeeth (532ms)
✓ future and squeeth (1309ms)
multiple sub-vaults
✓ future (873ms)
✓ squeeth (863ms)
✓ future and squeeth (1144ms)
getPositionValue
ETH price becomes high
✓ 1 long squeeth
✓ 1 short squeeth
✓ 1 long future
✓ 1 short future
✓ 1 long squeeth and 0.2 short future (49ms)
✓ 1 short squeeth and 1 long future (46ms)
✓ 1 short squeeth and 1 long future in different sub-vaults (246ms)
funding fee
✓ 1 long squeeth and positive funding fee
✓ 1 short squeeth and positive funding fee
✓ 1 long future and positive funding fee (41ms)
✓ 1 short future and positive funding fee (52ms)
✓ 1 long future and negative funding fee
✓ 1 short future and negative funding fee
getMinCollateralToAddPosition

```
    ✓ get min collateral of the vault
  USDCs are deposited
    ✓ get min collateral of the vault which has positions
  updateUsdcPosition
     single sub-vault
    ✓ more collateral required (39ms)
    ✓ there is excess collateral (95ms)
  multiple sub-vault
    ✓ more collateral required (49ms)
    ✓ there is excess collateral (99ms)
  checkVaultIsLiquidatable
     single sub-vault
    ✓ returns true if position value is less than min collateral
    ✓ returns true if position value is less than min collateral
    ✓ returns false if position value is greater than min collateral
  multiple sub-vault
    ✓ returns true if position value is less than min collateral
    ✓ returns false if position value is greater than min collateral
  setInsolvencyFlagIfNeeded
     single sub-vault
    ✓ reverts if there are positions
    ✓ vault is not insolvent
    ✓ vault is insolvent
  multiple sub-vault
    ✓ vault is not insolvent (41ms)
    ✓ vault is insolvent (43ms)
  decreaseLiquidationReward
    ✓ reward is 0.2 of MinCollateral
    ✓ reward is equal to usdcPosition
Contract: VaultNFT
  init
    ✓ set perpetual market
    ✓ reverts if not deployer
    ✓ reverts if address is 0
  mintNFT
    ✓ mint NFT
    ✓ reverts if not Perpetual Market
  370 passing (2m)
```

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by the Zokyo team

As part of our work assisting Predy team in verifying the correctness of their contract code, our team was responsible for writing integration tests using Truffle testing framework.

Tests were based on the functionality of the code, as well as review of the Predy contract requirements for details about issuance amounts and how the system handles these.

```
Contract: FeePool
    ✓ Owner should not withdraw if zero amount (49ms)
    ✓ Owner should not send profit if zero amount (20ms)
Contract: FlashHedge
    ✓ Should not swap if zero amounts (30ms)
Contract: PerpetualMarket
  Deployment requirements
    ✓ Deployer should not deploy the Perpetual Market if quote asset zero address (43ms)
    ✓ Deployer should not deploy the Perpetual Market if underlying asset zero address (34ms)
    ✓ Deployer should not deploy the Perpetual Market if fee recipient zero address (35ms)
Contract: PerpetualMarketCore
    ✓ Owner should not set a perpetual market if zero address (25ms)
    ✓ Owner should not set a perpetual market twice (27ms)
    ✓ Should not initialize a pool via a market if nonzero total supply (152ms)
    ✓ Should not deposit before an initialization of a pool (121ms)
    ✓ Should not get trade price if zero index price because of too small spot price (96ms)
    ✓ Should not get trade price if zero underling price (54ms)

  12 passing (9s)
```

Also, it needs to be mentioned, that the original code has a significant original coverage with testing scenarios provided by the Predy team. All of them were also carefully checked by the auditors team.

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	Uncovered Lines
contracts/	99.5	94.94	100		
FeePool.sol	100	100	100		
FlashHedge.sol	100	87.5	100		
PerpetualMarket.sol	100	97.22	100		
PerpetualMarketCore.sol	99.22	95.45	100		
VaultNFT.sol	100	100	100		
contracts/base/	100	75	90.91		
BaseFlashSwap.sol	100	66.67	87.5		
BaseLiquidityPool.sol	100	100	100		
All files	**99.01**	**92.07**	**99.4**		

We are grateful to have been given the opportunity to work with the Predy team.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that the Predy team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.