Report

v. 3.0

Customer

Hunch.finance

Smart Contract Audit

Hunch.finance

16th January 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog **6**

2 Introduction **7**

3 Project scope **8**

4 Methodology **9**

5 Our findings **10**

6 Critical Issues **11**
CVF-1. FIXED 11

7 Major Issues **12**
CVF-3. FIXED 12
CVF-5. FIXED 12
CVF-7. INFO 13
CVF-8. FIXED 13
CVF-9. FIXED 14
CVF-10. FIXED 14
CVF-12. INFO 14
CVF-14. FIXED 15
CVF-15. FIXED 15
CVF-16. FIXED 15
CVF-17. INFO 16
CVF-18. FIXED 16
CVF-19. FIXED 16
CVF-21. FIXED 17

8 Moderate Issues **18**
CVF-2. INFO 18
CVF-4. INFO 18
CVF-6. INFO 19
CVF-11. INFO 19
CVF-13. INFO 19
CVF-20. INFO 20
CVF-22. FIXED 20
CVF-23. FIXED 20
CVF-24. FIXED 21
CVF-25. FIXED 21
CVF-26. FIXED 21
CVF-27. FIXED 22
CVF-28. FIXED 22
CVF-29. FIXED 22

9 Minor Issues **23**
CVF-30. FIXED . . . 23
CVF-31. FIXED . . . 23
CVF-32. FIXED . . . 23
CVF-33. FIXED . . . 24
CVF-34. FIXED . . . 24
CVF-35. INFO . . . 25
CVF-36. FIXED . . . 25
CVF-37. FIXED . . . 25
CVF-38. FIXED . . . 26
CVF-39. FIXED . . . 26
CVF-40. FIXED . . . 26
CVF-41. FIXED . . . 26
CVF-42. FIXED . . . 27
CVF-43. FIXED . . . 27
CVF-44. FIXED . . . 27
CVF-45. FIXED . . . 27
CVF-46. FIXED . . . 28
CVF-47. FIXED . . . 28
CVF-48. INFO . . . 29
CVF-49. FIXED . . . 30
CVF-50. FIXED . . . 30
CVF-51. INFO . . . 31
CVF-52. FIXED . . . 32
CVF-53. FIXED . . . 32
CVF-54. FIXED . . . 32
CVF-55. INFO . . . 33
CVF-56. INFO . . . 33
CVF-57. FIXED . . . 34
CVF-58. FIXED . . . 34
CVF-59. FIXED . . . 34
CVF-60. INFO . . . 35
CVF-61. FIXED . . . 35
CVF-62. FIXED . . . 35
CVF-63. INFO . . . 36
CVF-64. FIXED . . . 36
CVF-65. FIXED . . . 36
CVF-66. INFO . . . 37
CVF-67. FIXED . . . 37
CVF-68. FIXED . . . 37
CVF-69. FIXED . . . 38
CVF-70. FIXED . . . 38
CVF-71. INFO . . . 39
CVF-72. INFO . . . 40
CVF-73. INFO . . . 40
CVF-74. INFO . . . 40

CVF-75. INFO . . . 41
CVF-76. INFO . . . 41
CVF-77. FIXED . . . 41
CVF-78. FIXED . . . 41
CVF-79. FIXED . . . 42
CVF-80. FIXED . . . 42
CVF-81. FIXED . . . 42
CVF-82. INFO . . . 43
CVF-83. INFO . . . 44
CVF-84. FIXED . . . 45
CVF-85. FIXED . . . 45
CVF-86. FIXED . . . 46
CVF-87. FIXED . . . 46
CVF-88. FIXED . . . 46
CVF-89. FIXED . . . 47
CVF-90. FIXED . . . 47
CVF-91. INFO . . . 48
CVF-92. FIXED . . . 48
CVF-93. FIXED . . . 48
CVF-94. FIXED . . . 49
CVF-95. FIXED . . . 49
CVF-96. FIXED . . . 49
CVF-97. INFO . . . 50
CVF-98. FIXED . . . 50
CVF-99. FIXED . . . 51
CVF-100. FIXED . . . 51
CVF-101. FIXED . . . 51
CVF-102. FIXED . . . 52
CVF-103. FIXED . . . 52
CVF-104. INFO . . . 52
CVF-105. FIXED . . . 53
CVF-106. FIXED . . . 53
CVF-107. FIXED . . . 53
CVF-108. INFO . . . 54
CVF-109. INFO . . . 54
CVF-110. FIXED . . . 54
CVF-111. INFO . . . 55
CVF-112. FIXED . . . 55
CVF-113. FIXED . . . 55
CVF-114. INFO . . . 56
CVF-115. FIXED . . . 56
CVF-116. FIXED . . . 56
CVF-117. INFO . . . 57
CVF-118. FIXED . . . 57
CVF-119. FIXED . . . 57
CVF-120. FIXED . . . 58

CVF-121. FIXED 58
CVF-122. INFO 59
CVF-123. FIXED 59
CVF-124. FIXED 60
CVF-125. FIXED 60
CVF-126. FIXED 61
CVF-127. FIXED 61
CVF-128. FIXED 62
CVF-129. FIXED 62
CVF-130. FIXED 63
CVF-131. FIXED 63

1 Changelog

#	Date	Author	Description
0.1	27.12.22	A. Zveryanskaya	Initial Draft
0.2	28.12.22	A. Zveryanskaya	Minor revision
1.0	28.12.22	A. Zveryanskaya	Release
1.1	12.01.23	A. Zveryanskaya	CVF-20 Client comment reduced
1.2	12.01.23	A. Zveryanskaya	CVF-35, CVF-48, CVF-63, CVF-71, CVF-97 Client comment fixed
1.3	12.01.23	A. Zveryanskaya	CVF-51 typo fixed
1.4	12.01.23	A. Zveryanskaya	CVF-89, CVF-99 status changed
1.5	12.01.23	A. Zveryanskaya	CVF-72, typo fixed
1.6	12.01.23	A. Zveryanskaya	CVF-75, CVF-76 comment changed
1.7	12.01.23	A. Zveryanskaya	CVF-91, typo fixed
2.0	12.01.23	A. Zveryanskaya	Release
2.1	16.01.23	A. Zveryanskaya	”Fix Repository” link up-dated
3.0	16.01.23	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

3 Project scope

We were asked to review:

- Original Repository
- Fix Repository

Files:

/			
	HunchInvestments.sol	MultiplierLib.sol	PositionsVault.sol
	UniswapHelper.sol		
interfaces/			
	IUniswapHelper.sol	ITicketFundsProvider.sol	IPositionsVault.sol
	IHunchInvestments.sol	IHunchInvestments Management.sol	
interfaces/games/			
	IHunchGame.sol	IFlippeningGame Management.sol	IFlippeningGame.sol
games/			
	FlippeningGame.sol	HunchGame.sol	

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

5 Our findings

We found 1 critical, 14 major, and a few less important issues. All identified Critical issues have been fixed.

Issues	Active	Fixed
Severity		
Critical	**0**	**1**
Major	**3**	**11**
Moderate	**6**	**8**
Minor	**26**	**76**

Fixed 96 out of 131 issues

6 Critical Issues

CVF-1. FIXED

- **Category** Flaw
- **Source** MultiplierLib.sol

Description These loops ignore the last values.

```
43 for (uint256 i = 0; i < _values.length - 1; i++) {
```

```
49 for (uint256 i = 0; i < _values.length - 1; i++) {
```

7 Major Issues

CVF-3. FIXED

- **Category** Suboptimal
- **Source** MultiplierLib.sol

Recommendation These formulas could be merged into one: ((x - x[i-1])y[i] + (x[i] - x])y[i-1])(x[i] - x[i-1])

```
22 return _yValues[firstBiggerIndex - 1] + (_x - _xValues[
      ↪ firstBiggerIndex - 1]) * (_yValues[firstBiggerIndex] -
      ↪ _yValues[firstBiggerIndex - 1]) / (_xValues[firstBiggerIndex]
      ↪ - _xValues[firstBiggerIndex - 1]);
```

```
24 return _yValues[firstBiggerIndex - 1] - (_x - _xValues[
      ↪ firstBiggerIndex - 1]) * (_yValues[firstBiggerIndex - 1] -
      ↪ _yValues[firstBiggerIndex]) / (_xValues[firstBiggerIndex] -
      ↪ _xValues[firstBiggerIndex - 1]);
```

CVF-5. FIXED

- **Category** Suboptimal
- **Source** MultiplierLib.sol

Description This condition is always true as "_values[i]" is "uint256".

Recommendation Consider removing this check.

```
31 require(_values[i] >= 0, "A multiplier value cannot be <0");
```

CVF-7. INFO

- **Category** Suboptimal
- **Source** FlippeningGame.sol

Recommendation Consider packing several values in a single storage slot to save space.

Client Comment *Won't fix. We target layer 2, gas is not as important and prefer redability.*

```
67 uint256[] public amountMultiplierXValues = [0.01 ether, 0.1 ether, 1
   ↪  ether, 2 ether, 4 ether, 10 ether];
   uint256[] public amountMultiplierYValues = [10 * 1000, 20 * 1000, 25
   ↪  * 1000, 30 * 1000, 35 * 1000, 40 * 1000];
```

```
70 uint256[] public ratioMultiplierXValues = [1, 500, 1000, 1500, 2000,
   ↪  2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000,
   ↪ 7500, 8000, 8500, 9000, 9500];
   uint256[] public ratioMultiplierYValues = [1000 * 10000, 300 *
   ↪ 10000, 100 * 10000, 80 * 10000, 50 * 10000, 25 * 10000, 14 *
   ↪ 10000, 10 * 10000, 7 * 10000, 55 * 1000,
    5 * 10000, 45 * 1000, 4 * 10000, 35 * 1000, 3 * 10000, 25 *
       ↪ 1000, 2 * 10000, 15 * 1000, 1 * 10000, 1000];
```

```
74 uint256[] public proximityMultiplierXValues = [SECONDS_PER_HOUR, 4 *
   ↪  SECONDS_PER_HOUR, 12 * SECONDS_PER_HOUR, SECONDS_PER_DAY, 2 *
   ↪  SECONDS_PER_DAY, 3 * SECONDS_PER_DAY,
    4 * SECONDS_PER_DAY, 5 * SECONDS_PER_DAY, 6 * SECONDS_PER_DAY, 7
       ↪  * SECONDS_PER_DAY];
   uint256[] public proximityMultiplierYValues = [25 * 10000, 16 *
   ↪ 10000, 12 * 10000, 9 * 10000, 6 * 10000, 4 * 10000, 3 * 10000,
   ↪  2 * 10000, 13 * 1000, 10000];
```

CVF-8. FIXED

- **Category** Overflow/Underflow
- **Source** FlippeningGame.sol

Description This line would underflow in case block.timestamp < CLAIM_WIN_PEROID.

Recommendation Consider checking as: block.timestamp - flippingDate < CLAIM_WIN_PERIOD.

```
163 require(block.timestamp - CLAIM_WIN_PERIOD < flippeningDate, "Too
   ↪ late");
```

CVF-9. FIXED

- **Category** Suboptimal
- **Source** FlippeningGame.sol

Description The value “notCollectedTicketIds.length” is read from the storage on every loop iteration.

Recommendation Consider reading once before the loop.

```
209 while (nonCollectedTicketIdIndex < notCollectedTicketIds.length &&
    ↪ collectCallsDone <= _maxPositionsToCollect) {
```

CVF-10. FIXED

- **Category** Suboptimal
- **Source** FlippeningGame.sol

Recommendation This check could be simplified. Just check whether the number of uncollected tickets is less than “_maxPositionsToCollect” and adjust the “_maxPositionsToCollected” before the loop if necessary.

```
209 while (nonCollectedTicketIdIndex < notCollectedTicketIds.length &&
    ↪ collectCallsDone <= _maxPositionsToCollect) {
```

CVF-12. INFO

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Recommendation Consider packing several values in a single storage slot to save space.

Client Comment *Won’t fix. We target layer 2, gas is not as important and prefer redability.*

```
44 uint256[] public TIME_MULTIPLIER_X_VALUES = [0, 1 days, 5 days, 30
    ↪ days, 160 weeks];
uint256[] public TIME_MULTIPLIER_Y_VALUES = [10000, 13000, 20000,
    ↪ 25000, 57400];
```

CVF-14. FIXED

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description The ”supportedTokens.length” value is read from the storage on every loop iteration.

Recommendation Consider reading once before the loop.

```
170 for (uint256 i = 0; i < supportedTokens.length; i++) {
```

CVF-15. FIXED

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Recommendation As all elements of the ”supportedTokens” array are unique, the loop should be terminated once the token is found.

```
171 if (token == supportedTokens[i]) {
```

CVF-16. FIXED

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Recommendation Consider using the staked position timestamp instead of its owner in this check to avoid extra storage read, as the timestamp would anyway be needed later.

```
181 require(stakedPositions[_tokenId].owner != address(0), "Not staked")
      ↪ ;
```

CVF-17. INFO

- **Category** Procedural
- **Source** HunchInvestments.sol

Description Here all the staked position attributes are read from the storage while only some of them are actually used.

Recommendation Consider reading only the necessary attributes.

Client Comment *Won't fix. We target layer 2 so gas not that critical, and prefer code readability*

189
```
StakeInfo memory stakedPosition = stakedPositions[_tokenId];
```

CVF-18. FIXED

- **Category** Flaw
- **Source** HunchInvestments.sol

Description Multiplication after division could lead to precision loss.

Recommendation Consider doing divisions after multiplications.

212
```
return uniswapHelper.getSpotPrice(factory.getPool(_token,
   ↪ wethAddress, _poolFee), _token0 == wethAddress) *
    _amount / 10 ** uniswapHelper.PRECISION_DECIMALS() * (
       ↪ MAX_POOL_FEE_PERCENTAGE - _poolFee) /
       ↪ MAX_POOL_FEE_PERCENTAGE;
```

CVF-19. FIXED

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description The value "games.length" is read from the storage on every loop iteration.

Recommendation Consider reading once before the loop.

350
```
for (uint256 i = 0; i < games.length; i++) {
```

372
```
for (uint256 i = 0; i < games.length; i++) {
```

382
```
for (uint256 i = 0; i < games.length; i++) {
```

CVF-21. FIXED

- **Category** Procedural
- **Source** IFlippeningGame.sol

Recommendation The “tokenId” parameters should be indexed.

```
5  event BuyETHTicket(address indexed account, uint256 ticketId,
       ↪ uint256 flippeningDate, uint256 marketCapRatio,

7  event BuyPositionETHTicket(address indexed account, uint256 ticketId
       ↪ , uint256 flippeningDate, uint256 marketCapRatio,

10 event BuyPositionTicket(address indexed account, uint256 ticketId,
       ↪ uint256 flippeningDate, uint256 marketCapRatio,

14 event ClaimWin(address indexed account, uint256 ticketId, uint256
       ↪ multipliedAmount, uint256 ratioMultiplier,

16 event CollectRewards(uint256 ticketId, uint256 positionTokenId,
       ↪ uint256 ethAmount);
   event ClaimReward(address indexed account, uint256 ticketId, uint256
       ↪  reward,
```

8 Moderate Issues

CVF-2. INFO

- **Category** Suboptimal
- **Source** MultiplierLib.sol

Description Linear search is inefficient.

Recommendation Consider using binary search instead.

Client Comment *Won't fix (n <= 10 at most).*

```
9 for (uint256 i = 0; i < _xValues.length; i++) {
```

CVF-4. INFO

- **Category** Overflow/Underflow
- **Source** MultiplierLib.sol

Description Phantom overflow is possible here.

Recommendation Consider using the "muldiv" function.

Client Comment *Won't fix (values are known in advance, no risk of overflowing).*

```
22 return _yValues[firstBiggerIndex - 1] + (_x - _xValues[
   ↪ firstBiggerIndex - 1]) * (_yValues[firstBiggerIndex] -
   ↪ _yValues[firstBiggerIndex - 1]) / (_xValues[firstBiggerIndex]
   ↪ - _xValues[firstBiggerIndex - 1]);
```

```
24 return _yValues[firstBiggerIndex - 1] - (_x - _xValues[
   ↪ firstBiggerIndex - 1]) * (_yValues[firstBiggerIndex - 1] -
   ↪ _yValues[firstBiggerIndex]) / (_xValues[firstBiggerIndex] -
   ↪ _xValues[firstBiggerIndex - 1]);
```

CVF-6. INFO

- **Category** Suboptimal
- **Source** MultiplierLib.sol

Description Performing this check on every loop iteration is suboptimal.

Recommendation Consider checking once and then implementing two separate loops.

Client Comment *Won't fix (check is of a parameter and not from storage, and this way it's more readable).*

```
33 if (isAccending) {
```

CVF-11. INFO

- **Category** Suboptimal
- **Source** FlippeningGame.sol

Recommendation Deleting the collected ticket IDs would refund some gas.

Client Comment *Won't fix: tickets will become collectibles, we want to keep their info forever.*

```
210 uint256 ticketId = notCollectedTicketIds[nonCollectedTicketIdIndex];
```

CVF-13. INFO

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description Should be a call to the "__Ownable_init_unchained" function.

Client Comment *Won't fix (they are the same function, and there is no double initialization issue).*

```
75 OwnableUpgradeable.__Ownable_init();
```

CVF-20. INFO

- **Category** Overflow/Underflow
- **Source** HunchGame.sol

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type while some intermediary calculation overflows.

Recommendation Consider using the “muldiv” function or some other ways that prevents phantom overflows.

Client Comment *Won’t fix (amount to overflow is nearly impossible).*

```
129 treasuryAmount = (_ethAmount * TREASURY_PERCENTAGE) / MAX_PERCENTAGE
      ↪ ;
```

CVF-22. FIXED

- **Category** Flaw
- **Source** FlippeningGame.sol

Description There are no checks that the passed arrays are not empty. Empty arrays could break the contract.

Recommendation Consider adding appropriate checks.

```
287 function setAmountMultiplier(uint256[] calldata _amounts, uint256[]
      ↪ calldata _multipliers) external override onlyOwner {
```

```
297 function setRatioMultiplier(uint256[] calldata _ratios, uint256[]
      ↪ calldata _multipliers) external override onlyOwner {
```

```
308 function setProximityMultiplier(uint256[] calldata _timeDiffs,
      ↪ uint256[] calldata _multipliers) external override onlyOwner {
```

CVF-23. FIXED

- **Category** Suboptimal
- **Source** PositionsVault.sol

Recommendation The value ”owners[_tokenId]” should be deleted here.

```
41 owner = owners[_tokenId];
```

CVF-24. FIXED

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Recommendation This loops wouldn't be necessary if the "isTokenSupported" mapping would be replaced with a mapping that stores indexes of all the supported tokens.

```
170 for (uint256 i = 0; i < supportedTokens.length; i++) {
```

CVF-25. FIXED

- **Category** Unclear behavior
- **Source** HunchInvestments.sol

Description This line should only be executed if the token was found in the "supportedTokens" array.

```
177 IERC20Upgradeable(token).safeApprove(address(router), 0);
```

CVF-26. FIXED

- **Category** Flaw
- **Source** HunchInvestments.sol

Description The returned value is ignored.

Recommendation Consider using the "safeTransfer" function or explicitly checking the returned value.

```
244 IERC20Upgradeable(token0).transfer(msg.sender, token0Fees);
    IERC20Upgradeable(token1).transfer(msg.sender, token1Fees);
```

```
283     IERC20Upgradeable(_token0).transfer(msg.sender, locals.
          ↪ collectedToken0Fees);
        IERC20Upgradeable(_token1).transfer(msg.sender, locals.
          ↪ collectedToken1Fees);
```

CVF-27. FIXED

- **Category** Unclear behavior
- **Source** HunchInvestments.sol

Description This line should be executed only when ”success” is true. Otherwise, an attempt to decode the returned data could revert.

```
327 (ethAmount) = abi.decode(returnData, (uint256));
```

CVF-28. FIXED

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description Linear search doesn’t scale.

Recommendation Consider maintaining a separate mapping of active games.

```
350 for (uint256 i = 0; i < games.length; i++) {
```

CVF-29. FIXED

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description Linear search doesn’t scale.

Recommendation Consider storing a mapping from game IDs to their indexed.

```
372 for (uint256 i = 0; i < games.length; i++) {
```

```
382 for (uint256 i = 0; i < games.length; i++) {
```

9 Minor Issues

CVF-30. FIXED

- **Category** Procedural
- **Source** MultiplierLib.sol

Description Compiler version requirement “^0.8” looks unusual and inconsistent with other files.

Recommendation Consider specifying as “^0.8.0”. Also relevanf for: FlippeningGame.sol, PositionsVault.sol, HunchInvestments.sol, HunchGame.sol.

```
2 pragma solidity ^0.8;
```

CVF-31. FIXED

- **Category** Suboptimal
- **Source** MultiplierLib.sol

Recommendation This constant is redundant. Just use “1 days” instead.

```
5 uint256 public constant SECONDS_PER_DAY = 3600 * 24;
```

CVF-32. FIXED

- **Category** Suboptimal
- **Source** MultiplierLib.sol

Description The value “_xValues.length” is read from the storage several times.

Recommendation Consider reading once and reusing.

```
8  uint256 firstBiggerIndex = _xValues.length;
   for (uint256 i = 0; i < _xValues.length; i++) {
16 if (firstBiggerIndex == _xValues.length) {
```

CVF-33. FIXED

- **Category** Suboptimal
- **Source** MultiplierLib.sol

Description The values "_xValues[firstBiggerIndex - 1]" and "_xValues[firstBiggerIndex]" were already read from the storage in the loop above.

Recommendation Consider reusing.

```
22 return _yValues[firstBiggerIndex - 1] + (_x - _xValues[
   ↪ firstBiggerIndex - 1]) * (_yValues[firstBiggerIndex] -
   ↪ _yValues[firstBiggerIndex - 1]) / (_xValues[firstBiggerIndex]
   ↪ - _xValues[firstBiggerIndex - 1]);
```

```
24 return _yValues[firstBiggerIndex - 1] - (_x - _xValues[
   ↪ firstBiggerIndex - 1]) * (_yValues[firstBiggerIndex - 1] -
   ↪ _yValues[firstBiggerIndex]) / (_xValues[firstBiggerIndex] -
   ↪ _xValues[firstBiggerIndex - 1]);
```

CVF-34. FIXED

- **Category** Suboptimal
- **Source** MultiplierLib.sol

Description The value "_yValues[firstBiggerIndex - 1]" is read from the storage twice.

Recommendation Consider reading once and reusing.

```
22 return _yValues[firstBiggerIndex - 1] + (_x - _xValues[
   ↪ firstBiggerIndex - 1]) * (_yValues[firstBiggerIndex] -
   ↪ _yValues[firstBiggerIndex - 1]) / (_xValues[firstBiggerIndex]
   ↪ - _xValues[firstBiggerIndex - 1]);
```

```
24 return _yValues[firstBiggerIndex - 1] - (_x - _xValues[
   ↪ firstBiggerIndex - 1]) * (_yValues[firstBiggerIndex - 1] -
   ↪ _yValues[firstBiggerIndex]) / (_xValues[firstBiggerIndex] -
   ↪ _xValues[firstBiggerIndex - 1]);
```

CVF-35. INFO

- **Category** Suboptimal
- **Source** MultiplierLib.sol

Description String error messages are suboptimal.

Recommendation Consider using named errors instead.

Client Comment *Won't fix. We prefer to keep the code more readable and compact as contract size and deploy gas are not an issue.*

```
31 require(_values[i] >= 0, "A multiplier value cannot be <0");
```

```
34         require(_values[i] <= _values[i + 1], "Not accending");
```

```
36         require(_values[i] >= _values[i + 1], "Not descending");
```

```
44 require (_values[i] <= 7 * SECONDS_PER_DAY, "A proximity multiplier
   ↪ cannot be greater than a week");
```

```
50 require (_values[i] < 10000, "A ratio multiplier cannot be ge 10000
   ↪ (100%)");
```

CVF-36. FIXED

- **Category** Documentation
- **Source** MultiplierLib.sol

Recommendation Should be 'ascending'.

```
33 if (isAccending) {
         require(_values[i] <= _values[i + 1], "Not accending");
```

CVF-37. FIXED

- **Category** Bad datatype
- **Source** MultiplierLib.sol

Recommendation The value 7 * SECOND_PER_DAY" should be a named constant.

```
44 require (_values[i] <= 7 * SECONDS_PER_DAY, "A proximity multiplier
   ↪ cannot be greater than a week");
```

CVF-38. FIXED

- **Category** Bad datatype
- **Source** MultiplierLib.sol

Recommendation The value “10000” should be a named constant.

```
50 require (_values[i] < 10000, "A ratio multiplier cannot be ge 10000
   ↪ (100%)");
```

CVF-39. FIXED

- **Category** Documentation
- **Source** FlippeningGame.sol

Description The number format of this field is unclear.

Recommendation Consider documenting.

```
20 uint256 marketCapRatio;
```

CVF-40. FIXED

- **Category** Suboptimal
- **Source** FlippeningGame.sol

Recommendation This constant is redundant. Just use “1 hours” instead.

```
28 uint256 public constant SECONDS_PER_HOUR = 3600;
```

CVF-41. FIXED

- **Category** Suboptimal
- **Source** FlippeningGame.sol

Recommendation This constant is redundant. Just use “1 days” instead.

```
29 uint256 public constant SECONDS_PER_DAY = 3600 * 24;
```

CVF-42. FIXED

- **Category** Suboptimal
- **Source** FlippeningGame.sol

Recommendation This initialization would be cheaper if “BETS_STATE” would be zero.

```
52 uint256 public state = BETS_STATE;
```

CVF-43. FIXED

- **Category** Bad datatype
- **Source** FlippeningGame.sol

Recommendation The type of this variable should be “IWETH9”.

```
60 address public wethAddress;
```

CVF-44. FIXED

- **Category** Readability
- **Source** FlippeningGame.sol

Recommendation These values could be rendered as “10e3”, “20e3” etc.

```
68 uint256[] public amountMultiplierYValues = [10 * 1000, 20 * 1000, 25
   ↪  * 1000, 30 * 1000, 35 * 1000, 40 * 1000];
```

CVF-45. FIXED

- **Category** Readability
- **Source** FlippeningGame.sol

Recommendation These values could be rendered as “1000e4”, “300e4” etc”.

```
71 uint256[] public ratioMultiplierYValues = [1000 * 10000, 300 *
   ↪ 10000, 100 * 10000, 80 * 10000, 50 * 10000, 25 * 10000, 14 *
   ↪ 10000, 10 * 10000, 7 * 10000, 55 * 1000,
    5 * 10000, 45 * 1000, 4 * 10000, 35 * 1000, 3 * 10000, 25 *
       ↪ 1000, 2 * 10000, 15 * 1000, 1 * 10000, 1000];
```

CVF-46. FIXED

- **Category** Readability
- **Source** FlippeningGame.sol

Recommendation These values could be rendered as “1 hours”, “4 hours”, “2 days” etc.

```
74 uint256[] public proximityMultiplierXValues = [SECONDS_PER_HOUR, 4 *
   ↪  SECONDS_PER_HOUR, 12 * SECONDS_PER_HOUR, SECONDS_PER_DAY, 2 *
   ↪  SECONDS_PER_DAY, 3 * SECONDS_PER_DAY,
```

CVF-47. FIXED

- **Category** Readability
- **Source** FlippeningGame.sol

Recommendation These values could be rendered as “25e4”, “16e4” etc.

```
76 uint256[] public proximityMultiplierYValues = [25 * 10000, 16 *
   ↪ 10000, 12 * 10000, 9 * 10000, 6 * 10000, 4 * 10000, 3 * 10000,
   ↪  2 * 10000, 13 * 1000, 10000];
```

CVF-48. INFO

- **Category** Suboptimal
- **Source** FlippeningGame.sol

Description String error messages are inefficient.

Recommendation Consider using named errors instead.

Client Comment *Won't fix. We prefer to keep the code more readable and compact as contract size and deploy gas are not an issue.*

```
79  require(!isCanceled, "Game canceled");
```

```
105         require(msg.value >= amountMultiplierXValues[0], "Not enough
               ↪  ETH");
```

```
148         require(state == BETS_STATE, "Already flipped");
```

```
152         require(ethMarketCap > btcMarketCap, "No flippening");
```

```
161         require(ownerOf(_ticketId) == msg.sender, "Not allowed");
            require(state == CLAIM_STATE, "Not allowed");
            require(block.timestamp - CLAIM_WIN_PERIOD < flippeningDate,
               ↪  "Too late");
```

```
166         require(betInfo.date > 0, "No ticket");
```

```
171         require(!betInfo.claimedWin, "Already claimed");
```

```
173         require(betInfo.date >= flippeningDate -
               ↪ FLIPPENING_WIN_MIN_DIST_PERIOD && betInfo.date <=
               ↪ flippeningDate + FLIPPENING_WIN_MIN_DIST_PERIOD, "
               ↪ Losing bet");
```

```
202         require(_maxPositionsToCollect > 0, "Max positions must be
               ↪ non-zero");
```

```
204         require(nextNonCollectedTicketIdIndex <
               ↪ notCollectedTicketIds.length, "Collection already done
               ↪ ");
```

```
239         require(ownerOf(_ticketId) == msg.sender, "Not allowed");
240         require(canClaimReward(), "Not allowed");
```

(243, 248, 260, 283, 288, 298, 309, 362, 269, 276, 381, 385)

CVF-49. FIXED

- **Category** Bad datatype
- **Source** FlippeningGame.sol

Recommendation The type of the "_wethAddress" argument should be "IWETH9".

```
83 constructor(AggregatorV3Interface _ethMarketCapOracle,
      ↪ AggregatorV3Interface _btcMarketCapOracle,
      ↪ ITicketFundsProvider _ticketFundsProvider, ISwapRouter _router
      ↪ , address _wethAddress, address _treasury)
```

CVF-50. FIXED

- **Category** Suboptimal
- **Source** FlippeningGame.sol

Description These checks are redundant as they basically validate constant values.

Recommendation Consider removing these checks.

```
92         MultiplierLib.validateOrder(amountMultiplierXValues, true);
           MultiplierLib.validateOrder(amountMultiplierYValues, true);

95         MultiplierLib.validateOrder(ratioMultiplierXValues, true);
           MultiplierLib.validateRatio(ratioMultiplierXValues);
           MultiplierLib.validateOrder(ratioMultiplierYValues, false);

99         MultiplierLib.validateOrder(proximityMultiplierXValues, true
              ↪ );
100        MultiplierLib.validateProximity(proximityMultiplierXValues);
           MultiplierLib.validateOrder(proximityMultiplierYValues,
              ↪ false);
```

CVF-51. INFO

- **Category** Overflow/Underflow
- **Source** FlippeningGame.sol

Description Phantom overflow is possible here.

Recommendation Consider using the “muldiv” function.

Client Comment *Won’t fix: amount can’t realistically be so large that an overflow will occur with any multiplier value. Same for marketCapRatio (which comes from an oracle so is even smaller), as well as for token price.*

```
181     multipliedAmount = multipliedAmount * ratioMultiplier /
           ↪ ONE_MULTIPLIER;
```

```
183     multipliedAmount = multipliedAmount * distanceMultiplier /
           ↪ ONE_MULTIPLIER;
```

```
197 _amount, _amount * _tokenPrice / 10 ** ERC20(_token).decimals()));
```

```
227     findersFee = ethCollected * FINDERS_FEE_PERCENTAGE /
           ↪ MAX_PERCENTAGE;
```

```
266             reward = fundsInfo.multipliedAmount * totalReward /
                   ↪ totalClaimedAmount;
```

```
322             multipliedETHAmount = _ethAmount * multiplier /
                   ↪ ONE_MULTIPLIER;
```

```
353     strength = strength * ratioMultiplier / ONE_MULTIPLIER;
```

```
357             strength = strength * proximityMultiplier /
                   ↪ ONE_MULTIPLIER;
```

```
369     require(uint256(ethMarketCap) * MAX_PERCENTAGE / uint256(
           ↪ btcMarketCap) <= MAX_ALLOWED_RATIO, "Flippening too close"
           ↪ );
```

```
371     marketCapRatio = uint256(ethMarketCap) * MAX_PERCENTAGE /
           ↪ uint256(btcMarketCap);
```

CVF-52. FIXED

- **Category** Bad datatype
- **Source** FlippeningGame.sol

Recommendation The type of the “_token” argument should be “IERC20”.

193
```
function convertFunds(address _token, uint256 _amount, uint256
    ↪ _tokenPrice) external override onlyOwner notCanceled returns (
    ↪ uint256 ethAmount) {
```

CVF-53. FIXED

- **Category** Suboptimal
- **Source** FlippeningGame.sol

Description In ERC20 the “decimals” property is used by UI to render token amounts in human-friendly format. Using this property in smart contracts is discouraged.

Recommendation Consider treating all token amounts as integers.

197
```
_amount, _amount * _tokenPrice / 10 ** ERC20(_token).decimals()));
```

CVF-54. FIXED

- **Category** Suboptimal
- **Source** FlippeningGame.sol

Description Using the “transfer” function is discouraged.

Recommendation Consider using “call” instead.

234
```
payable(msg.sender).transfer(findersFee);
payable(treasury).transfer(treasuryAmount);
```

272
```
        payable(msg.sender).transfer(reward);
```

CVF-55. INFO

- **Category** Procedural
- **Source** FlippeningGame.sol

Recommendation This check should be done earlier.

Client Comment *Won't fix: Don't see a reason why, all checks before are of ticket existence and other correction tests, this is just one of those tests, better done after ticket is valid and exists, to see it wasn't claimed.*

```
248 require(!betInfo.claimedReward, "Already rewarded");
```

CVF-56. INFO

- **Category** Suboptimal
- **Source** FlippeningGame.sol

Description These functions should emit some events.

```
278 function setNonAlpha() external override onlyOwner notCanceled {
```

```
282 function cancelGame() external override onlyOwner notCanceled {
```

```
287 function setAmountMultiplier(uint256[] calldata _amounts, uint256[]
    ↪ calldata _multipliers) external override onlyOwner {
```

```
297 function setRatioMultiplier(uint256[] calldata _ratios, uint256[]
    ↪ calldata _multipliers) external override onlyOwner {
```

```
308 function setProximityMultiplier(uint256[] calldata _timeDiffs,
    ↪ uint256[] calldata _multipliers) external override onlyOwner {
```

CVF-57. FIXED

- **Category** Suboptimal
- **Source** FlippeningGame.sol

Description The expression “uint256(ethMarketCap) * MAX_PERCENTAGE / uint256(btcMarketCap) ” is calculated twice.

Recommendation Consider calculating once and reusing.

```
369 require(uint256(ethMarketCap) * MAX_PERCENTAGE / uint256(
    ↪ btcMarketCap) <= MAX_ALLOWED_RATIO, "Flippening too close");
```

```
371 marketCapRatio = uint256(ethMarketCap) * MAX_PERCENTAGE / uint256(
    ↪ btcMarketCap);
```

CVF-58. FIXED

- **Category** Procedural
- **Source** UniswapHelper.sol

Description Specifying an unbounded compiler versions range is a bad practice, as one cannot guarantee compatibility with future major versions.

Recommendation Consider specifying as ”^0.7.0 || ^0.8.0”.

```
2 pragma solidity >=0.7.6;
```

CVF-59. FIXED

- **Category** Bad datatype
- **Source** UniswapHelper.sol

Recommendation The type of the ”_poolAddress” arguments should be ”IUniswapV3Pool”.

```
16 function getTWAPPrice(address _poolAddress, uint32 _interval, bool
    ↪ _isToken0ETH) external view override returns (uint256 price) {
```

```
29 function getSpotPrice(address _poolAddress, bool _isToken0ETH)
    ↪ external view override returns (uint256 price) {
```

CVF-60. INFO

- **Category** Unclear behavior
- **Source** UniswapHelper.sol

Description Strictly speaking this function calculates the exponent of an average logarithm of prices, rather than an average price. So the calculated value is more like a geometric mean, rather than arithmetic one. Probably not an issue.

Client Comment *Won't fix: We followed guidelines from UniswapV3 TWAP calculation, to calculate the mean tick, and then go from it to the price. You're right the mean is on price square, but as Unsiwap's recommendation, we do this as it easily implemented and recommended by them.*

```
16 function getTWAPPrice(address _poolAddress, uint32 _interval, bool
      ↪ _isToken0ETH) external view override returns (uint256 price) {
```

CVF-61. FIXED

- **Category** Documentation
- **Source** PositionsVault.sol

Description The semantics of keys in this mapping is unclear.

Recommendation Consider documenting.

```
11 mapping(uint256 => address) public owners;
```

CVF-62. FIXED

- **Category** Suboptimal
- **Source** PositionsVault.sol

Description These functions should emit some events.

```
25 function put(uint256 _tokenId) external override returns (address
      ↪ owner) {
```

```
32 function collect(uint256 _tokenId) external override returns (
      ↪ uint256 token0Fees, uint256 token1Fees) {
```

```
39 function release(uint256 _tokenId) external override returns (
      ↪ address owner) {
```

CVF-63. INFO

- **Category** Suboptimal
- **Source** PositionsVault.sol

Description String error messages are inefficient.

Recommendation Consider using named errors instead.

Client Comment *Won't fix. We prefer to keep the code more readable and compact as contract size and deploy gas are not an issue.*

```
26 require(msg.sender == operator, "Not allowed");
```

```
33 require(msg.sender == operator, "Not allowed");
```

```
40 require(msg.sender == operator, "Not allowed");
```

CVF-64. FIXED

- **Category** Bad datatype
- **Source** HunchInvestments.sol

Recommendation The type of this variable should be "IERC20[]".

```
49 address[] public supportedTokens;
```

CVF-65. FIXED

- **Category** Bad datatype
- **Source** HunchInvestments.sol

Recommendation The key type for this mapping should be "IERC20".

```
50 mapping(address => bool) public isTokenSupported;
```

CVF-66. INFO

- **Category** Procedural
- **Source** HunchInvestments.sol

Recommendation These variables should be declared as immutable.

Client Comment *Won't fix. Cannot be immutable as it is settable from initialzie as contract is upgradeable.*

```
54 IPositionsVault public positionsVault;
   IUniswapHelper public uniswapHelper;
   INonfungiblePositionManager public positionManager;
   ISwapRouter public router;
   IUniswapV3Factory public factory;

60 address public wethAddress;
```

CVF-67. FIXED

- **Category** Bad datatype
- **Source** HunchInvestments.sol

Recommendation The type of this variable should be "IWETH9".

```
60 address public wethAddress;
```

CVF-68. FIXED

- **Category** Procedural
- **Source** HunchInvestments.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
62 receive() external payable {

64 }
```

CVF-69. FIXED

- **Category** Bad datatype
- **Source** HunchInvestments.sol

Recommendation The type of the ”_wethAddress” should be ”IWETH9”.

```
66 function initialize(IPositionsVault _positionsVault, IUniswapHelper
   ↪ _uniswapHelper, INonfungiblePositionManager _positionManager,
   ↪ ISwapRouter _router, IUniswapV3Factory _factory, address
   ↪ _wethAddress) public initializer {
```

CVF-70. FIXED

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description These checks basically validate constants which is waste of gas.

Recommendation Consider removing these checks.

```
77 MultiplierLib.validateOrder(TIME_MULTIPLIER_X_VALUES, true);
   MultiplierLib.validateOrder(TIME_MULTIPLIER_Y_VALUES, true);
```

CVF-71. INFO

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description String error messages are inefficient.

Recommendation Consider using named errors.

Client Comment *Won't fix. We prefer to keep the code more readable and compact as contract size and deploy gas are not an issue.*

```
90  require(stakedPositions[_tokenId].gameId == 0 && stakedPositions[
       ↪ _tokenId].ticketId == 0, "Close ticket to unstake");
```

```
96  require(IERC721(address(game)).ownerOf(_ticketId) == position.owner,
       ↪  "Bad ticket id");
    require(position.owner == _owner, "Not allowed");
    require(position.gameId == game.gameId() && position.ticketId ==
       ↪ _ticketId, "Not allowed");
```

```
110 require(IERC721(address(game)).ownerOf(_ticketId) == position.owner,
       ↪  "Bad ticket id");
    require(position.gameId == game.gameId() && position.ticketId ==
       ↪ _ticketId, "Not allowed");
```

```
121 require(IERC721(address(game)).ownerOf(_ticketId) == _owner, "Bad
       ↪ ticket id");
```

```
128 require(IERC721(address(game)).ownerOf(_ticketId) == _owner, "Bad
       ↪ ticket id");
    require(position.owner == _owner, "Not allowed");
130 require(position.gameId == 0 && position.ticketId == 0, "Already
       ↪ gamed");
```

```
138 require(IERC721(address(game)).ownerOf(_ticketId) == _owner, "Bad
       ↪ ticket id");
    require(position.owner == _owner, "Not allowed");
140 require(position.gameId == game.gameId() && position.ticketId ==
       ↪ _ticketId, "Not allowed");
```

```
161 require(!isTokenSupported[token], "Already supported");
```

```
181 require(stakedPositions[_tokenId].owner != address(0), "Not staked")
       ↪ ;
```

(218, 220, 222, 234, 271, 356, 363, 366, 384)

CVF-72. INFO

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description This check makes the ”owner” argument redundant.

Client Comment *Won't fix: It's not. This is called by game, sending the _owner as the msg.sender, and so guarantees the game is not using its permissions to get funds from a position not initiated by the owner via closePositionTicket or claimWin. It is possible if done by getFunds, but that can only happen for losing tickets and is ok.*

```
97 require(position.owner == _owner, "Not allowed");
```

CVF-73. INFO

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description This check makes the ”_ticketId” argument redundant.

Client Comment *Won't fix: Makes sure the ticket being used is indeed the one that was staked with the position.*

```
98 require(position.gameId == game.gameId() && position.ticketId ==
     ↪ _ticketId, "Not allowed");
```

CVF-74. INFO

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description This check makes the ”_owner” argument redundant.

Client Comment *Won't fix: Not redundant, as it makes sure the caller to the game (passed as _owner) is indeed the owner of the ticket. Though theoretically the game checks this as well, this is a double safety check.*

```
121 require(IERC721(address(game)).ownerOf(_ticketId) == _owner, "Bad
      ↪ ticket id");
```

CVF-75. INFO

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description This check makes the ”_owner” argument redundant.

Client Comment *Same as CVF-74.*

```
129 require(position.owner == _owner, "Not allowed");
```

CVF-76. INFO

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description This check makes the ”_owner” argument redundant.

Client Comment *Same as CVF-74.*

```
139 require(position.owner == _owner, "Not allowed");
```

CVF-77. FIXED

- **Category** Bad datatype
- **Source** HunchInvestments.sol

Recommendation The return type should be ”IERC20[]”.

```
156 function getSupportedTokens() view external returns (address[]
    ↪ memory) {
```

CVF-78. FIXED

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description This function doesn’t scale.

Recommendation Consider implementing an ability to query ranges of the ”supported-Tokens” array.

```
156 function getSupportedTokens() view external returns (address[]
    ↪ memory) {
```

CVF-79. FIXED

- **Category** Bad datatype
- **Source** HunchInvestments.sol

Description Converting a token to ”IERC20Upgradeable” is confusing and tokens don’t have to all be upgradeable.

Recommendation Consider converting to ”IERC20” instead.

```
165 IERC20Upgradeable(token).safeApprove(address(router), type(uint256).
    ↪ max);
```

```
177 IERC20Upgradeable(token).safeApprove(address(router), 0);
```

CVF-80. FIXED

- **Category** Unclear behavior
- **Source** HunchInvestments.sol

Description The rest of the function shouldn’t be executed if ”isTokenSupported[token]” wasn’t true.

```
169 isTokenSupported[token] = false;
```

CVF-81. FIXED

- **Category** Documentation
- **Source** HunchInvestments.sol

Description The multiplier number format is unclear.

Recommendation Consider documenting.

```
180 function multiplierOf(uint256 _tokenId) override view public returns
    ↪  (uint256 multiplier) {
```

```
188 function getTimeMultipliedFees(uint256 _tokenId, uint256 _token0Fees
    ↪ , uint256 _token1Fees) external view override returns (uint256
    ↪  totalTimeMultipliedETHFees, uint256 timeMultiplier, uint256
    ↪ totalETHFees) {
```

CVF-82. INFO

- **Category** Procedural
- **Source** HunchInvestments.sol

Recommendation The expression ”10 ** uniswapHelper.PRECISION_DECIMALS()” could be precomputed.

Client Comment *Won’t fix: theoretically UniswapHelper can change its precision, so we prefer not to pre-compute in case of a change.*

213
```
_amount / 10 ** uniswapHelper.PRECISION_DECIMALS() * (
    ↪ MAX_POOL_FEE_PERCENTAGE - _poolFee) / MAX_POOL_FEE_PERCENTAGE;
```

321
```
        _amount, _amount * _price / 10 ** uniswapHelper.
            ↪ PRECISION_DECIMALS()));
```

326
```
            _amount, _amount * _price / 10 ** uniswapHelper.
                ↪ PRECISION_DECIMALS())));
```

CVF-83. INFO

- **Category** Overflow/Underflow
- **Source** HunchInvestments.sol

Description Phantom overflow is possible here.

Recommendation Consider using the ”muldiv” function.

Client Comment *Won't fix: as all amounts are based on actual fees or eth amount, it is extremely radical for them to be very large, and as price's are token's prices in ETH, multipliers are maximum 1e4, as well as MAX_PERCENTAGE based multiplications, it is not realistic to even come close to overflowing uint256's 76+ digits. Therefore, for code redability, we prefer not to use mulDiv.*

```
212     return uniswapHelper.getSpotPrice(factory.getPool(_token,
           ↪ wethAddress, _poolFee), _token0 == wethAddress) *
            _amount / 10 ** uniswapHelper.PRECISION_DECIMALS() * (
               ↪ MAX_POOL_FEE_PERCENTAGE - _poolFee) /
               ↪ MAX_POOL_FEE_PERCENTAGE;
```

```
301     multipliedToken0ETHAmount = _ETHToken0Fees *
           ↪ _collectedToken0Fees / (_collectedToken0Fees +
           ↪ _baseToken0Fees);
```

```
305     multipliedToken1ETHAmount = _ETHToken1Fees *
           ↪ _collectedToken1Fees / (_collectedToken1Fees +
           ↪ _baseToken1Fees);
```

```
309 timeMultipliedETHAmount = (multipliedToken0ETHAmount +
       ↪ multipliedToken1ETHAmount) * timeMultiplier / ONE_MULTIPLIER +
```

```
321                 _amount, _amount * _price / 10 ** uniswapHelper.
                       ↪ PRECISION_DECIMALS()));
```

```
326                   _amount, _amount * _price / 10 ** uniswapHelper.
                         ↪ PRECISION_DECIMALS())));
```

```
340     slippagedPrice = tokenPrice - tokenPrice * _slippage /
           ↪ MAX_PERCENTAGE;
```

CVF-84. FIXED

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description The expression ”stakedPositions[_tokenId]” is calculated several times.

Recommendation Consider calculating once and reusing.

```
218 require(stakedPositions[_tokenId].owner == address(0), "Already
       ↪ staked");
```

```
224 stakedPositions[_tokenId] = StakeInfo(_owner, block.timestamp, 0, 0,
       ↪  _gameId, _ticketId);
```

```
230     stakedPositions[_tokenId].token0Fees, stakedPositions[_tokenId].
           ↪ token1Fees, _gameId, _ticketId);
```

CVF-85. FIXED

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Recommendation If ”wethAddress” would be declared as internal, it would be possible to optimize this condition by comparing with ”wethAddress” first and then checking whether the other token is supported.

```
222 require(isTokenSupported[token0] && token1 == wethAddress ||
       ↪ isTokenSupported[token1] && token0 == wethAddress, "Pair not
       ↪ supported");
```

CVF-86. FIXED

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description Here the staked position attributes just written to the storage are read back.

Recommendation Consider reusing the written values.

```
227 (stakedPositions[_tokenId].token0Fees, stakedPositions[_tokenId].
      ↪ token1Fees) = positionsVault.collect(_tokenId);
```

```
230     stakedPositions[_tokenId].token0Fees, stakedPositions[_tokenId].
          ↪ token1Fees, _gameId, _ticketId);
```

CVF-87. FIXED

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description The expression ”stakedPositions[_tokenId]” is calculated several times.

Recommendation Consider calculating once and reusing.

```
234 require(stakedPositions[_tokenId].timestamp > 0, "Not staked");
    require(stakedPositions[_tokenId].owner == _owner, "Not owner");
```

```
237 uint256 token0Fees = stakedPositions[_tokenId].token0Fees;
    uint256 token1Fees = stakedPositions[_tokenId].token1Fees;
```

```
240 delete stakedPositions[_tokenId];
```

CVF-88. FIXED

- **Category** Bad datatype
- **Source** HunchInvestments.sol

Recommendation The type of the token arguments should be ”IERC20”.

```
261 function _getFunds(uint256 _tokenId, StakeInfo memory _position,
      ↪ address _token0, address _token1, uint24 _poolFee, uint256
      ↪ _token0ETHPrice, uint256 _token1ETHPrice, bool _shouldRevert)
```

CVF-89. FIXED

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description Using the ”transfer” function is discouraged.

Recommendation Consider using the ”call” function instead.

```
280 payable(msg.sender).transfer(ethAmount);
```

CVF-90. FIXED

- **Category** Bad datatype
- **Source** HunchInvestments.sol

Recommendation The type of the ”_token” argument should be ”IERC20”.

```
313 function exactInputSwap(bool _shouldRevert, address _token, uint24
    ↪ _poolFee, uint256 _amount, uint256 _price) private returns (
    ↪ bool success, uint256 ethAmount) {
```

```
337 function getSlippagedPrice(address _token, uint24 _poolFee, uint256
    ↪ _slippage, address _token0) private view returns (uint256
    ↪ slippagedPrice) {
```

CVF-91. INFO

- **Category** Suboptimal
- **Source** HunchInvestments.sol

Description These two code chunks basically do the same and differ only in how they handle errors.

Recommendation Consider always using the "call" function and handle unsuccess differently depending on the "_shouldRevert" value.

Client Comment *Won't fix: We won't get inner failure string easily, better keep it this way.*

```
320 ethAmount = router.exactInput(ISwapRouter.ExactInputParams(abi.
    ↪ encodePacked(_token, _poolFee, wethAddress), address(this),
    ↪ block.timestamp,
     _amount, _amount * _price / 10 ** uniswapHelper.
        ↪ PRECISION_DECIMALS()));
```

```
324 (success, returnData) =
      address(router).call(abi.encodeWithSelector(router.exactInput.
         ↪ selector, ISwapRouter.ExactInputParams(abi.encodePacked(
         ↪ _token, _poolFee, wethAddress), address(this), block.
         ↪ timestamp,
          _amount, _amount * _price / 10 ** uniswapHelper.
             ↪ PRECISION_DECIMALS())));
```

CVF-92. FIXED

- **Category** Bad datatype
- **Source** HunchInvestments.sol

Recommendation The type of the token returned values should be "IERC20".

```
344 function requireValidPosition(uint256 _tokenId) private view returns
    ↪  (address token0, address token1, uint24 poolFee) {
```

CVF-93. FIXED

- **Category** Bad datatype
- **Source** HunchGame.sol

Recommendation The type of this variable should be "address payable".

```
28 address public treasury;
```

CVF-94. FIXED

- **Category** Procedural
- **Source** HunchGame.sol

Description Unlike the “setTreasury” function, constructor allows zero treasury address.

Recommendation Consider making the constructor behavior consistent with the setter behavior.

```
34 address _treasury
```

CVF-95. FIXED

- **Category** Bad datatype
- **Source** HunchGame.sol

Recommendation The type of the “_treasury” arguments should be “address payable”.

```
34     address _treasury
```

```
44 function setTreasury(address _treasury) external override onlyOwner
   ↪ {
```

CVF-96. FIXED

- **Category** Documentation
- **Source** HunchGame.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
40 receive() external payable {
```

```
42 }
```

CVF-97. INFO

- **Category** Suboptimal
- **Source** HunchGame.sol

Description String error messages are inefficient.

Recommendation Consider using named error instead.

Client Comment *Won't fix. We prefer to keep the code more readable and compact as contract size and deploy gas are not an issue.*

```
45 require(_treasury != address(0), "Bad address");
```

```
52 require(_ethAmount > 0, "Amount not positive");
```

```
61 require(ethAmount > 0, "Fees not positive");
```

```
69     require(owner == msg.sender, "Not allowed");
```

```
83 require(ownerOf(_ticketId) == msg.sender, "Not allowed");
```

CVF-98. FIXED

- **Category** Suboptimal
- **Source** HunchGame.sol

Description The expression "tickets[_ticketId]" is calculated several times.

Recommendation Consider calculating once and reusing.

```
86     tickets[_ticketId].tokenId, msg.sender, _ticketId,
          ↪ _token0ETHPrice, _token1ETHPrice);
```

```
88 tickets[_ticketId].tokenId = 0;
   tickets[_ticketId].amount = ethAmount - treasuryAmount;
```

```
94 tickets[_ticketId].multipliedAmount = multipliedAmount;
   TicketFundsInfo memory ticket =  tickets[_ticketId];
```

CVF-99. FIXED

- **Category** Suboptimal
- **Source** HunchGame.sol

Description Using the “transfer” function is discouraged.

Recommendation Consider using “call” instead.

97 `payable(treasury).transfer(treasuryAmount);`

118 `payable(treasury).transfer(treasuryAmount);`

CVF-100. FIXED

- **Category** Documentation
- **Source** HunchGame.sol

Description The number format of the “multiplier” returned value is unclear.

Recommendation Consider documenting.

104 `returns (uint256 multipliedETHAmount, uint256 multiplier);`

CVF-101. FIXED

- **Category** Suboptimal
- **Source** HunchGame.sol

Description Here values just written into the storage are read back.

Recommendation Consider reusing the written values.

116 `ticket = tickets[ticketId];`

CVF-102. FIXED

- **Category** Suboptimal
- **Source** HunchGame.sol

Description The “nextTicketInfo” value is read from the storage twice.

Recommendation Consider reading once: ticketId = nextTicketId++; _safeMint(msg.sender, ticketId);

```
122 ticketId = nextTicketId;
```

```
124 nextTicketId++;
```

CVF-103. FIXED

- **Category** Procedural
- **Source** HunchGame.sol

Recommendation Brackets are redundant here.

```
129 treasuryAmount = (_ethAmount * TREASURY_PERCENTAGE) / MAX_PERCENTAGE
    ↪ ;
```

CVF-104. INFO

- **Category** Suboptimal
- **Source** HunchGame.sol

Recommendation This formula could be simplified as: treasuryAmount = _ethAmount / 20;

Client Comment *Won’t fix. We prefer to live like this for readability, as percentage out of MAX_PERCENTAGE repeats in code in a few places.*

```
129 treasuryAmount = (_ethAmount * TREASURY_PERCENTAGE) / MAX_PERCENTAGE
    ↪ ;
```

CVF-105. FIXED

- **Category** Procedural
- **Source** IUniswapHelper.sol

Description Specifying an unbounded compiler versions range is discouraged, as one cannot guarantee compatibility with future major versions.

Recommendation Consider specifying as ”^0.7.0 || ^ 0.8.0”.

```
2 pragma solidity >=0.7.6;
```

CVF-106. FIXED

- **Category** Documentation
- **Source** IUniswapHelper.sol

Description The price number format is unclear.

Recommendation Consider documenting.

```
5 function getTWAPPrice(address poolAddress, uint32 interval, bool
     ↪ isToken0ETH) external view returns (uint256 price);
  function getSpotPrice(address poolAddress, bool isToken0ETH)
     ↪ external view returns (uint256 price);
```

CVF-107. FIXED

- **Category** Bad datatype
- **Source** IUniswapHelper.sol

Recommendation The type of the ”poolAddress” arguments could be more specific.

```
5 function getTWAPPrice(address poolAddress, uint32 interval, bool
     ↪ isToken0ETH) external view returns (uint256 price);
  function getSpotPrice(address poolAddress, bool isToken0ETH)
     ↪ external view returns (uint256 price);
```

CVF-108. INFO

- **Category** Suboptimal
- **Source** IUniswapHelper.sol

Recommendation Turning this function into a constant could save gas.

Client Comment *Won't fix. We target layer 2 and want the precision decimal of prices to be expose for HunchInvestments.*

```
8 function PRECISION_DECIMALS() external view returns (uint256);
```

CVF-109. INFO

- **Category** Suboptimal
- **Source** IPositionsVault.sol

Description This interface misses gettter functions.

Recommendation Consider adding appropriate getters.

Client Comment *Won't fix. All relevant info is public but without an interface, so can be queried from outside the blockchain easily. If needed by another contract, a special interface can be created then.*

```
4 interface IPositionsVault {
```

CVF-110. FIXED

- **Category** Unclear behavior
- **Source** IPositionsVault.sol

Recommendation These functions should emit some events and these events should be declared in this interface.

```
5 function put(uint256 tokenId) external returns (address owner);
  function collect(uint256 tokenId) external returns (uint256
     ↪ token0Fees, uint256 token1Fees);
  function release(uint256 tokenId) external returns (address owner);
```

CVF-111. INFO

- **Category** Suboptimal
- **Source** IHunchInvestmentsManagement.sol

Description This interface misses getter functions, so there is no way to observe the protocol state.

Recommendation Consider adding appropriate getters.

Client Comment *Won't fix. All relevant info is public but without an interface, so can be queried from outside the blockchain easily. If needed by another contract, a special interface can be created then.*

```
6 interface IHunchInvestmentsManagement {
```

CVF-112. FIXED

- **Category** Unclear behavior
- **Source** IHunchInvestmentsManagement.sol

Recommendation These functions should emit some events and these events should be declared in this interface.

```
7 function addGame(IHunchGame game) external;
  function removeGame(IHunchGame game) external;
```

```
10 function addSupportedToken(address token) external;
   function removeSupportedToken(address token) external;
```

CVF-113. FIXED

- **Category** Bad datatype
- **Source** IHunchInvestmentsManagement.sol

Recommendation The type of the "token" arguments should be "IERC20".

```
10 function addSupportedToken(address token) external;
   function removeSupportedToken(address token) external;
```

CVF-114. INFO

- **Category** Suboptimal
- **Source** IHunchInvestments.sol

Description This interface misses getter functions, so there is not way to observe the current stakes.

Recommendation Consider adding appropriate getters.

Client Comment *Won't fix. It is declared in ITicketFundsProvider as it is needed for the communication between contracts, and implemented using public override.*

```
4 interface IHunchInvestments {
```

CVF-115. FIXED

- **Category** Procedural
- **Source** IHunchInvestments.sol

Recommendation The "tokenId" parameters should be indexed.

```
5 event Stake(address indexed owner, uint256 tokenId, address token0,
    ↪ address token1,
```

```
7 event Unstake(address indexed owner, uint256 tokenId);
```

CVF-116. FIXED

- **Category** Bad datatype
- **Source** IHunchInvestments.sol

Recommendation The type of the token parameters should be "IERC20".

```
5 event Stake(address indexed owner, uint256 tokenId, address token0,
    ↪ address token1,
```

CVF-117. INFO

- **Category** Bad naming
- **Source** IHunchGame.sol

Recommendation Events are usually named via nouns.

Client Comment *Won't fix. We prefer readability over event names as noun, to match the action done as is an option as well in some big projects (like gmx for example).*

```
5 event ClosePositionTicket(address indexed account, uint256 ticketId,
    ↪  uint256 positionTokenId,
```

CVF-118. FIXED

- **Category** Procedural
- **Source** IHunchGame.sol

Recommendation The "ticketId" and "positionTokenId" parameters should be indexed.

```
5 event ClosePositionTicket(address indexed account, uint256 ticketId,
    ↪  uint256 positionTokenId,
```

CVF-119. FIXED

- **Category** Documentation
- **Source** IHunchGame.sol

Description The number format of fees and multiplier parameters is unclear.

Recommendation Consider documenting.

```
6 uint256 ethAmount, uint256 fees, uint256 ticketETHAmount, uint256
    ↪ ticketMultipliedETHAmount, uint256 amountMultiplier);
```

CVF-120. FIXED

- **Category** Documentation
- **Source** IFlippeningGame.sol

Description The number format of the fees and multiplier parameters is unclear.

Recommendation Consider documenting.

```
6         uint256 ethAmount, uint256 fees, uint256 ticketETHAmount,
             ↪ uint256 ticketMultipliedETHAmount, uint256
             ↪ amountMultiplier);

8         uint256 positionTokenId, uint256 ethAmount, uint256 fees,
             ↪ uint256 ticketETHAmount,
          uint256 ticketMultipliedETHAmount, uint256 amountMultiplier)
             ↪ ;

14    event ClaimWin(address indexed account, uint256 ticketId, uint256
         ↪ multipliedAmount, uint256 ratioMultiplier,
              uint256 distanceMultiplier);
```

CVF-121. FIXED

- **Category** Documentation
- **Source** IFlippeningGame.sol

Description The number format of ratio parameters is unclear.

Recommendation Consider documenting.

```
5     event BuyETHTicket(address indexed account, uint256 ticketId,
         ↪ uint256 flippeningDate, uint256 marketCapRatio,

7     event BuyPositionETHTicket(address indexed account, uint256 ticketId
         ↪ , uint256 flippeningDate, uint256 marketCapRatio,

10    event BuyPositionTicket(address indexed account, uint256 ticketId,
         ↪ uint256 flippeningDate, uint256 marketCapRatio,

14    event ClaimWin(address indexed account, uint256 ticketId, uint256
         ↪ multipliedAmount, uint256 ratioMultiplier,
```

CVF-122. INFO

- **Category** Bad naming
- **Source** IFlippeningGame.sol

Recommendation Events are usually named via nouns.

Client Comment *Won't fix. We prefer readability over event names as noun, to match the action done as is an option as well in some big projects (like gmx for example).*

```
5  event BuyETHTicket(address indexed account, uint256 ticketId,
       ↪ uint256 flippeningDate, uint256 marketCapRatio,

7  event BuyPositionETHTicket(address indexed account, uint256 ticketId
       ↪ , uint256 flippeningDate, uint256 marketCapRatio,

10 event BuyPositionTicket(address indexed account, uint256 ticketId,
       ↪ uint256 flippeningDate, uint256 marketCapRatio,

14 event ClaimWin(address indexed account, uint256 ticketId, uint256
       ↪ multipliedAmount, uint256 ratioMultiplier,

16 event CollectRewards(uint256 ticketId, uint256 positionTokenId,
       ↪ uint256 ethAmount);
   event ClaimReward(address indexed account, uint256 ticketId, uint256
       ↪  reward,
```

CVF-123. FIXED

- **Category** Documentation
- **Source** IFlippeningGame.sol

Description The number format of price values is unclear.

Recommendation Consider documenting.

```
21 function buyPositionETHTicket(uint256 positionTokenId, uint256
       ↪ flippeningDate, uint256 token0ETHPrice, uint256 token1ETHPrice
       ↪ ) external returns (uint256 ticketId);

23 function closePositionTicket(uint256 ticketId, uint256
       ↪ token0ETHPrice, uint256 token1ETHPrice) external;

26 function claimWin(uint256 ticketId, uint256 token0ETHPrice, uint256
       ↪ token1ETHPrice) external;
```

CVF-124. FIXED

- **Category** Bad naming
- **Source** IFlippeningGame.sol

Description Unlike other functions, these functions have argument names with underscore ("_") prefix.

Recommendation Consider using a consistent naming schema across the code.

30
```
function getMultipliedETHAmount(uint256 _ethAmount, uint256 /*
   ↪ _ticketId*/) external view returns (uint256
   ↪ multipliedETHAmount, uint256 multiplier);
function getRatioMultiplier(uint256 _ratio) external view returns (
   ↪ uint256 multiplier);
function getFlippeningDistanceMultiplier(uint256 _betDate) external
   ↪ view returns (uint256);
```

CVF-125. FIXED

- **Category** Documentation
- **Source** IFlippeningGame.sol

Description The number format of ratio values is unclear.

Recommendation Consider documenting.

31
```
function getRatioMultiplier(uint256 _ratio) external view returns (
   ↪ uint256 multiplier);
```

34
```
function strengthOf(uint256 ticketId, uint256 token0Fees, uint256
   ↪ token1Fees) external view returns (uint256 strength, uint256
   ↪ amountMultiplier, uint256 ratioMultiplier, uint256
   ↪ proximityMultiplier);
```

CVF-126. FIXED

- **Category** Documentation
- **Source** IFlippeningGame.sol

Description The number format of fees, strength, and multiplier values is unclear.

Recommendation Consider documenting.

```
34 function strengthOf(uint256 ticketId, uint256 token0Fees, uint256
   ↪ token1Fees) external view returns (uint256 strength, uint256
   ↪ amountMultiplier, uint256 ratioMultiplier, uint256
   ↪ proximityMultiplier);
```

CVF-127. FIXED

- **Category** Unclear behavior
- **Source** IFlippeningGameManagement.sol

Recommendation These functions should emit some events and these events should be declared in this interface.

```
5  function setNonAlpha() external;
   function cancelGame() external;
```

```
8  function convertFunds(address token, uint256 amount, uint256
   ↪ tokenPrice) external returns (uint256 ethAmount);
```

```
10 function setAmountMultiplier(uint256[] calldata amounts, uint256[]
   ↪ calldata multipliers) external;
   function setRatioMultiplier(uint256[] calldata ratios, uint256[]
   ↪ calldata multipliers) external;
   function setProximityMultiplier(uint256[] calldata timeDiffs,
   ↪ uint256[] calldata multipliers) external;
```

CVF-128. FIXED

- **Category** Documentation
- **Source** IFlippeningGameManagement.sol

Description The number format of price and multiplier values is unclear.

Recommendation Consider documenting.

```
8  function convertFunds(address token, uint256 amount, uint256
       ↪ tokenPrice) external returns (uint256 ethAmount);

10 function setAmountMultiplier(uint256[] calldata amounts, uint256[]
       ↪ calldata multipliers) external;
   function setRatioMultiplier(uint256[] calldata ratios, uint256[]
       ↪ calldata multipliers) external;
   function setProximityMultiplier(uint256[] calldata timeDiffs,
       ↪ uint256[] calldata multipliers) external;
```

CVF-129. FIXED

- **Category** Procedural
- **Source** ITicketFundsProvider.sol

Recommendation The “tokenId” parameter should be indexed.

```
7  event Close(address indexed owner, uint256 tokenId, uint256
       ↪ ethAmount, uint256 timeMultipliedETHAmount,
```

CVF-130. FIXED

- **Category** Documentation
- **Source** ITicketFundsProvider.sol

Description The number format of price values is unclear.

Recommendation Consider documenting.

```
11 function getFunds(uint256 tokenId, address owner, uint256 ticketId,
      ↪ uint256 token0ETHPrice, uint256 token1ETHPrice) external
      ↪ returns (uint256 ethAmount, uint256 timeMultipliedETHAmount);
   function getFunds(uint256 tokenId, address owner, uint256
      ↪ token0ETHPrice, uint256 token1ETHPrice) external returns (
      ↪ uint256 ethAmount, uint256 timeMultipliedETHAmount);
   function getFundsWithoutPrices(uint256 tokenId, uint256 ticketId,
      ↪ uint256 slippage) external returns (uint256 ethAmount, uint256
      ↪  timeMultipliedETHAmount);
```

CVF-131. FIXED

- **Category** Unclear behavior
- **Source** ITicketFundsProvider.sol

Recommendation These functions should emit some events and these events should be declared in this interface.

```
15 function stakeWithTicket(uint256 tokenId, address owner, uint256
      ↪ ticketId) external;
   function updateTicketInfo(uint256 tokenId, address owner, uint256
      ↪ ticketId) external;
   function unstakeForOwner(uint256 tokenId, address owner, uint256
      ↪ ticketId) external;
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting