VIDMA

HEADS TAILS

SMART CONTRACT AUDIT

Project: HeadsTails
Date: September 14th, 2022

TABLE OF CONTENTS

SUMMARY

Vidma is pleased to present this audit report outlining our assessment of code, smart contracts, and other important audit insights and suggestions for management, developers, and users.

HeadsTails is a community-powered platform for betting on the predictions of real-world events using cryptocurrency. Every topic is a yes-or-no question. Users simply bet on whether they think the answer to the question is YES or NO. Once the event transpires, the entire pool of funds is split amongst those who bet on the winning side. Users receive rewards based on their stake portion in the winning pool.

Every question has a verifiable end date. HeadsTails must resolve the question and select the winning side within 7 days of the question end date. If this is not completed on time, users can freely unstake their tokens. Stakers are needed on both sides of the question for a market to resolve successfully. If a question ends with 0 stakers on either side, it is allowed to be canceled. Users can unstake their tokens then.

The owner of the contract has permission to answer the question once it is ended for staking and once it is answered contract fee and submitter fee are transferred.

During the audit process, the Vidma team found several issues, including those with critical severity. A detailed summary and the current state are displayed in the table below.

Severity of the issue	Total found	Resolved	Unresolved
Critical	1 issue	1 issue	0 issues
High	1 issue	1 issue	0 issues
Medium	2 issues	2 issues	0 issues
Low	3 issues	3 issues	0 issues
Informational	2 issues	2 issues	0 issues
Total	**9 issues**	**9 issues**	**0 issues**

After evaluating the findings in this report and the final state after fixes, the Vidma auditors can state that the contracts are fully operational and secure. Under the given circumstances, we set the following risk level:

Our auditors are evaluating the initial commit given for the scope of the audit and the last commit with the fixes. This approach helps us adequately and sequentially evaluate the quality of the code. Code style, optimization of the contracts, the number of issues, and risk level of the issues are all taken into consideration. The Vidma team has developed a transparent scoring system presented below.

Severity of the issue	Resolved	Unresolved
Critical	1	10
High	0.8	7
Medium	0.5	5
Low	0.2	0.5
Informational	0	0.1

Please note that the points are deducted out of 100 for each and every issue on the list of findings (according to the current status of the issue). Issues marked as "not valid" are not subject to point deduction.

Based on the **overall result of the audit**, the Vidma audit team grants the following score:

In addition to manual check and static analysis, the auditing team has conducted a number of integrated autotests to ensure the given codebase has an adequate performance and security level.

The test results and the coverage can be found in the accompanying section of this audit report.

Please be aware that this audit does not certify the definitive reliability and security level of the contract. This document describes all vulnerabilities, typos, performance issues, and security issues found by the Vidma audit team. If the code is still under development, we highly recommend running one more audit once the code is finalized.

SCOPE OF WORK

HEADS TAILS

HeadsTails is a community-driven platform for betting on the predictions of real-world events using cryptocurrency. Topics span a range of categories, including politics, world events, sports, science, public health, and more.

Within the scope of this audit, two independent auditors thoroughly investigated the given codebase and analyzed the overall security and performance of the smart contracts.

The audit was conducted from September 1st, 2022 to September 14th, 2022. The outcome is disclosed in this document.

The scope of work for the given audit consists of the following contract:

- HeadsTails.

The source code was taken from the following **source**:
https://github.com/headstailsxyz/heads-tails-smart-contract/blob/main/contracts

Initial commit submitted for the audit:
105814efd11227395c81122a70237a4532b513d1

Last commit reviewed by the auditing team:
db7f8d577e0fa418b1e5a8d0c40adb3a7c659232

As a reference to the contracts logic, business concept, and the expected behavior of the codebase, the HeadsTails team has provided the following documentation:
https://github.com/headstailsxyz/litepaper

WORKFLOW OF THE AUDITING PROCESS

Vidma audit team uses the most sophisticated and contemporary methods and well-developed techniques to ensure contracts are free of vulnerabilities and security risks. The overall workflow consists of the following phases:

Phase 1: The research phase

Research

After the Audit kick-off, our security team conducts research on the contract's logic and expected behavior of the audited contracts.

Documentation reading

Vidma auditors do a deep dive into your tech documentation with the aim of discovering all the behavior patterns of your codebase and analyzing the potential audit and testing scenarios.

The outcome

At this point, the Vidma auditors are ready to kick off the process. We set the auditing strategies and methods and are prepared to conduct the first audit part.

Phase 2: Manual part of the audit

Manual check

During the manual phase of the audit, the Vidma team manually looks through the code in order to find any security issues, typos, or discrepancies with the logic of the contract. The initial commit as stated in the agreement is taken into consideration.

Static analysis check

Static analysis tools are used to find any other vulnerabilities in smart contracts that were missed after a manual check.

The outcome

An interim report with the list of issues.

Phase 3: Testing part of the audit

Integration tests

Within the testing part, Vidma auditors run integration tests using the Truffle or Hardhat testing framework. The test coverage and the test results are inserted in the accompanying section of this audit report.

The outcome

Second interim report with the list of new issues found during the testing part of the audit process.

STRUCTURE AND ORGANIZATION OF THE FINDINGS

For simplicity in reviewing the findings in this report, Vidma auditors classify the findings in accordance with the severity level of the issues. (from most critical to least critical).

All issues are marked as “Resolved” or “Unresolved”, depending on if they have been fixed by HeadsTails or not. The issues with “Not Valid” status are left on the list of findings but are not eligible for the score points deduction.

The latest commit with the fixes reviewed by the auditors is indicated in the “Scope of Work” section of the report.

The Vidma team always provides a detailed description of the issues and recommendations on how to fix them.

Classification of found issues is graded according to 6 levels of severity described below:

Critical

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

Example: Underflow/overflow, precisions, locked funds.

High

The issue significantly affects the ability of the contract to compile or operate. These are potential security or operational issues.

Example: Compilation errors, pausing/unpausing of some functionality, a random value, recursion, the logic that can use all gas from block (too many iterations in the loop), no limitations for locking period, cooldown, arithmetic errors which can cause underflow, etc.

Medium

The issue slightly impacts the contract's ability to operate by slightly hindering its intended behavior.

Example: Absence of emergency withdrawal of funds, using assert for parameter sanitization.

Low

The issue doesn't contain operational or security risks, but are more related to optimization of the codebase.

Example: Unused variables, inappropriate function visibility (public instead of external), useless importing of SCs, misuse or disuse of constant and immutable, absent indexing of parameters in events, absent events to track important state changes, absence of getters for important variables, usage of string as a key instead of a hash, etc.

Informational

Are classified as every point that increases onboarding time and code reading, as well as the issues which have no impact on the contract's ability to operate.

Example: Code style, NatSpec, typos, license, refactoring, naming convention (or unclear naming), layout order, functions order, lack of any type of documentation.

MANUAL REPORT

Possibility of Reentrancy attack

Critical | MC – 01 | Resolved

In function *unstake()* two *if else* statements were merged. But, stakeInfo.processed is updated after BNB/token transfer it can lead to Reentrancy attack.

Recommendation:

Move stakeInfo.processed from L490 to L478 before the *if else* statement. Also consider adding *OpenZeppelin's ReentrancyGuard* to methods where BNB/custom tokens are transferred to prevent the possibility of Reentrancy attack.

The owner has a possibility to block user funds

High | MH – 01 | Resolved

Contact is designed in such a manner that if the project team didn't define the winning side for a settled answer period then users who participate in that question round should be able to withdraw their tokens. As the answer period can be changed by the contract owner to any positive uint256 value it can lead to blocking the user's possibility to withdraw funds from not answered questions.

Recommendation:

Consider the possibility to limit max answering time.

Unchecked call return value

Medium | MM – 01 | Resolved

The return value of a token transfer function is not checked.

Recommendation:

As the contract is designed to support different tokens added by the contract owner consider a possibility to use *safeTransfer*, *safeTransferFrom* from OpenZeppelin instead of transfer for all token transfers as it adds the possibility to safely operate with not standard tokens which don't have return value on transfers.

Discrepancies between documentation and actual functionality

Medium | MM – 02 | Resolved

According to documentation the minimum amount of submitter fee can be set to 0.1% but actually it can be settled to 0.01%.

Recommendation:

Change documentation in a proper way *or*/*and* modify the function to receive the expected behavior.

Lock pragma to a specific version

Low | ML – 01 | Resolved

At the current state for part of the contract pragma is set to version range ^0.8.9 It's best practice to lock the pragma to a specific version since not all the EVM compiler versions support all the features, especially the latest one which are kind of beta versions, so the intended behavior written in code might not be executed as expected.

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs.

Recommendation:

Consider locking pragma to a specific version.

Presence of unused event

Low | ML – 02 | Resolved

The code should not contain unused events if this is not justified by design.

Recommendation:

Consider removing unused event *WithDraw*.

Code optimization

Low | ML – 03 | Resolved

In the contract methods *unstake()* and *harvest()* used repeated *if else* statement

Recommendation:

Consider moving stakeInfo.processed before the *if else* statement and merge them together.

Boolean equality

Informational | MI – 01 | Resolved

Boolean constants can be used directly and do not need to be compared to true or false.

Recommendation:

Remove the equality to the boolean constant.

Style guide violation

Informational | MI – 02 | Resolved

Functions should be grouped according to their visibility and ordered in the following way:

- constructor;
- receive function (if exists);
- fallback function (if exists);
- external;
- public;
- internal;
- private.

Ordering helps readers identify which functions they can call and find the constructor and fallback definitions easier.

Also, variable *ANSWERING_PERIOD* is not in mixedCase.

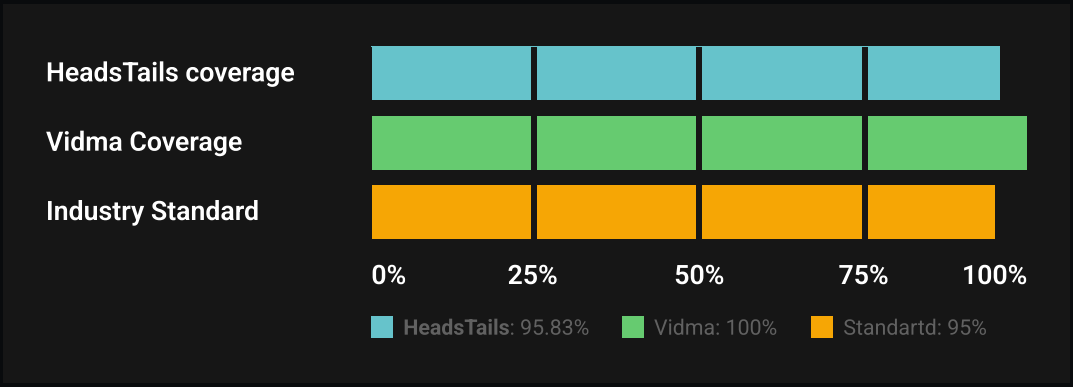
Recommendation:

Follow the official Solidity code style.

TEST RESULTS

To verify the security of contracts and the performance, a number of integration tests were carried out using the Truffle testing framework.

In this section, we provide both tests written by HeadsTails and tests written by Vidma auditors.



It is important to note that Vidma auditors do not modify, edit or add tests to the existing tests provided in the HeadsTails repository. We write totally separate tests with code coverage of a minimum of 95% to meet the industry standards.

Tests written by HeadsTails

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	95.83	83.19	93.75	96.02
HeadsTails.sol	95.83	83.19	93.75	96.02
All Files	**95.83**	**83.19**	**93.75**	**96.02**

Test Results

```
Contract: HeadsTails
  Deployment
    ✓ Should set the right owner (43ms)
  Checking `setContractFee(uint256 _contractFee) external onlyOwner`
    ✓ Checking `onlyOwner` (77ms)
    ✓ Contract fee should be a positive number (49ms)
    ✓ HeadsTails: Contract fee should be less than 30% (44ms)
    ✓ Run success: (update contractFee 500 to 1000) (70ms)
  Checking `setSubmitterFee(uint256 _submitterFee) external
  onlyOwner`
    ✓ Checking `onlyOwner`
    ✓ Submitter fee should be a positive number (50ms)
    ✓ Submitter fee should be less than 2%
    ✓ Run success: (update submitterFee 10 to 100)
  Checking `addStakeToken(address token, uint256 maxStakeAmount)
  external onlyOwner`
    ✓ Checking `onlyOwner`
    ✓ The zero address should not be a stake token (39ms)
    ✓ Token has been added already (56ms)
    ✓ Min stake amount should be a positive number
    ✓ Min stake amount should be smaller than max stake
      amount (89ms)
```

```
  ✓ Run success: (tokenList index equal to be
    stakeTokenCounter.current()) (192ms)
  ✓ Run success: (the token address of tokeList equal to
    be the param token address) (98ms)
  ✓ Run success: (the max amount of tokenList equal to
    be the param of max stake amount) (38ms)
  ✓ Run success: (stakeTokenIndices[stakeTokenCounter.current()]
    equal to be the token param address) (58ms)
  ✓ Run success: (check stakeTokenCounter.increment()) (73ms)
Checking `setMaxStakeAmount(address token, uint256
maxStakeAmount) external onlyOwner`
  ✓ Checking `onlyOwner` (41ms)
  ✓ The zero address should not be a stake token
  ✓ The token is not added as a supported token
  ✓ Max stake amount should be greater than min stake
    amount (111ms)
  ✓ Run success: (the max stake amount of tokenList equal
    to be as the param maxStakeAmount) (76ms)
  ✓ Run success: (check the item has been changed) (73ms)
Checking `setStakeTokenStatus(address token, bool status)
external onlyOwner`
  ✓ Checking `onlyOwner`
  ✓ The zero address should not be a stake token
  ✓ The token is not added as a supported token
  ✓ Run success (the status of stake token list equal to be as
    the param status) (94ms)
  ✓ Run success (check the correct item of token list has
    been changed) (78ms)
Checking `function createQuestion(address token, uint256 endDate,
string memory questionID, address submitter) external onlyOwner`
  ✓ Checking `onlyOwner` (48ms)
  ✓ Checking `The token is not enabled as a staked token.`
  ✓ Checking `End date should be greater than the current
    time.` (70ms)
  ✓ Checking `Empty string should not be added as a
    questionID.` (87ms)
  ✓ Run success: (set question contract fee as contact fee
    when question is created) (135ms)
  ✓ Run success: (set question submitter fee as submitter fee
    when question is created) (124ms)
```

```
    ✓ Run success: (set submitter address of question as zero
      address in the case of sumitter param address is
      zero address) (192ms)
    ✓ Run success: (set submitter address of question as param
      submitter address when question was created) (74ms)
    ✓ Run success: (set question item index as
      questionCounter.current()) (73ms)
    ✓ Run success: (check end date) (152ms)
    ✓ Run success: (questionCounter increment) (74ms)
Checking `function stakeToken(uint256 questionId,bool answer,
uint256 stakeAmount) external`
    ✓ Question has not started
    ✓ This function does not work on questions with the native
      token (76ms)
    ✓ Question has already been answered (242ms)
    ✓ Question has already been cancelled
    ✓ Staking is no longer allowed for this Question (38ms)
    ✓ Question has ended already
    ✓ Stake amount should be greater than the token min stake
      amount
    ✓ You cannot change the answer (89ms)
    ✓ Stake amount should be smaller than the available
      amount (99ms)
    ✓ Your token balance is insufficient for this stake (54ms)
    ✓ Your token allowance amount is insufficient for this
      stake (44ms)
    ✓ Run success: (question counter increment) (107ms)
    ✓ Run success: (set client list flag) (83ms)
    ✓ Run success: (set answer of stake info) (76ms)
    ✓ Run success: (set amount of stake info) (144ms)
    ✓ Run success: (set proceed as false in default) (78ms)
    ✓ Run success: (increase totalStakeAmount) (72ms)
    ✓ Run success: (full check) (89ms)
Checking `function stakeNativeToken(uint256 questionId,bool answer)
external payable`
    ✓ Question has not started
    ✓ This function does not work on questions with the native
      token (73ms)
    ✓ Question has already been answered (114ms)
    ✓ Question has already been cancelled (46ms)
    ✓ Staking is no longer allowed for this Question (40ms)
    ✓ Question has ended already
```

```
  ✓ Stake amount should be greater than the token min stake
    amount
  ✓ You cannot change the answer (81ms)
  ✓ Stake amount should be smaller than the available
    amount (91ms)
  ✓ Run success: Check native token balance in case of
    native token (66ms)
  ✓ Run success: Check stake amount in case of native
    token (65ms)
  ✓ Run success: (question counter increment) (59ms)
  ✓ Run success: (set client list flag) (61ms)
  ✓ Run success: (set answer of stake info) (70ms)
  ✓ Run success: (set amount of stake info) (62ms)
  ✓ Run success: (set proceed as false in default) (56ms)
  ✓ Run success: (increase totalStakeAmount) (61ms)
  ✓ Run success: (full check) (72ms)
Checking `unstake(uint256 questionId) external`
  ✓ Question has not started
  ✓ The End Date has not been reached for this question.
    Please check back later
  ✓ Question has been answered already (129ms)
  ✓ You can unstake if the question is not answered after
    1 week of the end date
  ✓ You have already unstaked (73ms)
  ✓ You have not staked tokens in this Question
  ✓ Run success: Check value in the case of native token (113ms)
  ✓ Run success: (set processed true)
  ✓ Run success: (transfer subitter fee) (41ms)
  ✓ Run success: (full check) (48ms)
Checking `harvest(uint256 questionId) external`
  ✓ Question has not started
  ✓ You have not staked tokens in this Question (68ms)
  ✓ Question has not been answered
  ✓ Your answer is wrong (62ms)
  ✓ You have already claimed your profits (96ms)
  ✓ Run success: Check native token (208ms)
  ✓ Run success: (set processed as true) (79ms)
  ✓ Run success: (submitter is the zero address) (428ms)
  ✓ Run success: (harvest token) (116ms)
Checking `function cancelQuestion(uint256 questionId) external`
  ✓ Question has not started
  ✓ Question has been answered (126ms)
```

```
    ✓ Question has been cancelled
    ✓ The End Date has not been reached for this question.
      Please check back later
    ✓ You can cancel the question which there are 0 stakers
      on either side (99ms)
    ✓ Run success (96ms)
  Checking `function answerQuestion(uint256 questionId,
  bool answer) external onlyOwner`
    ✓ Only onwer
    ✓ Question has not started
    ✓ Question has been answered (129ms)
    ✓ The End Date has not been reached for this question
    ✓ As 7 Days have passed after end date, its too late to
      answer this question
    ✓ Run success: (set anser of the item in questions
      list) (122ms)
    ✓ Run success: (set anwer flag as true of the item in
      question list) (120ms)
    ✓ Run success: (update total winning amount) (140ms)
    ✓ Run success: (update total commission amount) (140ms)
    ✓ Run success: (transfer submitter fee to submitter) (147ms)
    ✓ Run success: (submitter fee is 0 in case of zero
      address) (224ms)
    ✓ Run success: (submitter fee is zero in case of submitter
      is the zero address [native token]) (146ms)
    ✓ Run success: (transfer submitter fee to submitter in
      case of native token) (199ms)

  115 passing (41s)
```

Tests written by Vidma auditors

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	100.00	94.12	100.00	100.00
HeadsTails.sol	100.00	94.12	100.00	100.00
All Files	**100.00**	**94.12**	**100.00**	**100.00**

Test Results

```
Contract: HeadsTails
  Initialization
    ✓ should initialize question fee and answering period
      correct (85ms)
    ✓ should set owner correct
  setContractFee
    ✓ shouldn't set contract fee as 0 (54ms)
    ✓ shouldn't set contract fee bigger than 30%
    ✓ shouldn't set contract fee by not the owner
    ✓ should set contract fee correct
    ✓ should emit event correct (51ms)
  setSubmitterFee
    ✓ shouldn't set submitter fee as 0
    ✓ shouldn't set submitter fee bigger than 2% (79ms)
    ✓ shouldn't set submitter fee by not the owner
    ✓ should set submitter fee correct
    ✓ should emit event correct (44ms)
  setMaxStakeAmount
    ✓ shouldn't set max stake amount for zero token address (53ms)
    ✓ shouldn't set max stake amount for not supported token
    ✓ shouldn't set max stake amount less than min stake
      amount for the token (86ms)
```

```
    ✓ shouldn't set max stake amount by not the owner
    ✓ should set max stake amount correct (124ms)
    ✓ should emit event correct (114ms)
  setMinStakeAmount
    ✓ shouldn't set min stake amount for zero token address
    ✓ shouldn't set min stake amount for not supported token (49ms)
    ✓ shouldn't set min stake amount bigger than max stake
      amount for the token (108ms)
    ✓ shouldn't set min stake amount as 0
    ✓ shouldn't set min stake amount by not the owner
    ✓ should set min stake amount correct (73ms)
    ✓ should emit event correct (38ms)
  addStakeToken
    ✓ shouldn't add stake token with zero token address
    ✓ shouldn't add stake token twice (88ms)
    ✓ shouldn't add stake token with min stake amount equal 0
    ✓ shouldn't add stake token with min stake amount
      bigger than max
    ✓ shouldn't add stake token by not the owner
    ✓ should add stake token correct (60ms)
    ✓ should emit event correct
  setStakeTokenStatus
    ✓ shouldn't set stake token status for zero token address
    ✓ shouldn't set stake token status for not supported token
    ✓ shouldn't set stake token status by not the owner
    ✓ should set stake token status correct (91ms)
    ✓ should emit event correct (94ms)
  setStakeable
    ✓ shouldn't set stakeable status for the not started question
    ✓ shouldn't set stakeable status for the already answered
      question (203ms)
    ✓ shouldn't set stakeable status for the already ended
      question (143ms)
    ✓ shouldn't set stakeable status by not the owner
    ✓ should set stakeable status correct (93ms)
    ✓ should emit event correct (65ms)
  createQuestion
    ✓ shouldn't create question with not supported stake token (39ms)
    ✓ shouldn't create question with an invalid end date (43ms)
    ✓ shouldn't create question with an empty questionID (76ms)
    ✓ shouldn't create question by not the owner
    ✓ should create question correct (105ms)
```

```
    ✓ should create multiple questions correct (184ms)
    ✓ should emit event correct (61ms)
  stakeToken
    ✓ shouldn't stake for the not started question
    ✓ shouldn't stake if question supports native token (178ms)
    ✓ shouldn't stake if question is answered already (170ms)
    ✓ shouldn't stake if staking is not allowed for the
      question (80ms)
    ✓ shouldn't stake if question staking period is ended
      already (275ms)
    ✓ shouldn't stake too small amount (95ms)
    ✓ shouldn't stake too big amount (156ms)
    ✓ shouldn't stake if question is canceled (94ms)
    ✓ shouldn't stake if the user has an insufficient amount
      of the staked token (81ms)
    ✓ shouldn't stake if the contract has lack allowance of
      the user's staked token (79ms)
    ✓ shouldn't stake twice by the one user for the different
      answers (112ms)
    ✓ should stake twice for the same answer correct (191ms)
    ✓ should stake by multiple users correct (161ms)
    ✓ should emit event correct (112ms)
  stakeNativeToken
    ✓ shouldn't stake for the not started question
    ✓ shouldn't stake if question does not support native
      token (104ms)
    ✓ shouldn't stake if question is answered already (188ms)
    ✓ shouldn't stake if staking is not allowed for the
      question (90ms)
    ✓ shouldn't stake if question staking period is ended
      already (115ms)
    ✓ shouldn't stake too small amount (113ms)
    ✓ shouldn't stake too big amount (160ms)
    ✓ shouldn't stake if question is canceled (77ms)
    ✓ shouldn't stake twice by the one user for the different
      answers (159ms)
    ✓ should stake twice for the same answer correct (167ms)
    ✓ should stake by multiple users correct (144ms)
    ✓ should emit event correct (170ms)
  unstake
    ✓ shouldn't unstake from the not started question (66ms)
    ✓ shouldn't unstake if the question not ended (150ms)
```

```
  ✓ shouldn't unstake if question is answered already (137ms)
  ✓ shouldn't unstake if question is in answering period
    still (123ms)
  ✓ shouldn't unstake by not the staker (144ms)
  ✓ shouldn't unstake twice (172ms)
  ✓ should unstake native coin correct (235ms)
  ✓ should unstake ERC20 token correct (154ms)
  ✓ should emit event correct (155ms)
harvest
  ✓ shouldn't harvest not started question
  ✓ shouldn't harvest canceled question (161ms)
  ✓ shouldn't harvest not answered question (67ms)
  ✓ shouldn't harvest if user is not the staker (268ms)
  ✓ shouldn't harvest if user stake for the loss side (172ms)
  ✓ shouldn't harvest twice (224ms)
  ✓ should harvest native coin correct [no charge
    submitter fee] (315ms)
  ✓ should harvest ERC20 token correct [charge
    submitter fee] (402ms)
answerQuestion
  ✓ shouldn't answer not started question
  ✓ shouldn't answer question by not the owner
  ✓ shouldn't answer already answered question (151ms)
  ✓ shouldn't answer already canceled question (115ms)
  ✓ shouldn't answer not ended question (135ms)
  ✓ shouldn't answer question when answering period is
    over (138ms)
  ✓ shouldn't answer question if there is a lack of stakes
    on one of sides (121ms)
  ✓ should answer question correct [no charge submitter
    fee] (453ms)
  ✓ should answer question correct [charge submitter
    fee] (480ms)
cancelQuestion
  ✓ shouldn't cancel not started question (42ms)
  ✓ shouldn't cancel answered question (170ms)
  ✓ shouldn't cancel already canceled question (82ms)
  ✓ shouldn't cancel not ended question (63ms)
  ✓ shouldn't cancel questions with at least one stake on each
    side (162ms)
  ✓ should cancel question correct (86ms)
  ✓ should emit event correct (64ms)
```

```
  isTrueOrFalseCounterZero
    ✓ should check if there are any stakes on one of the sides (75ms)

110 passing (47s)
```

We are delighted to have a chance to work with the HeadsTails team and contribute to your company's success by reviewing and certifying the security of your smart contracts.

The statements made in this document should be interpreted neither as investment or legal advice, nor should its authors be held accountable for decisions made based on this document.

Vidma is a security audit company helping crypto companies ensure their code and products operate safely and as intended, enabling founders to sleep soundly at night. We specialize in auditing DeFi protocols, layer one protocols, and marketplace solutions. Our team consists of experienced and internationally trained specialists. Our company is based in Ukraine, known for its strong engineering, cryptography, and cybersecurity culture.

Website: **vidma.io**
Email: **security@vidma.io**

VIDMA