Hedgey Finance

Security Assessment

June 14th, 2021

For :
Hedgey Finance

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.

Overview

Project Summary

Project Name	Hedgey Finance
Description	Decentralized Finance Protocol
Platform	Ethereum; Solidity; Yul
Codebase	GitHub Repository
Commits	a65e14bf42ee26731ad47b35030d6588bdc875cd

Audit Summary

Delivery Date	June. 14th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Mar. 26, 2021 - June. 14, 2021

Vulnerability Summary

Total Issues	17
● **Total Critical**	0
● **Total Major**	7
● **Total Minor**	2
● **Total Informational**	7
● **Total Discussion**	1

Executive Summary

This report has been prepared for Hedgey Finance smart contract to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

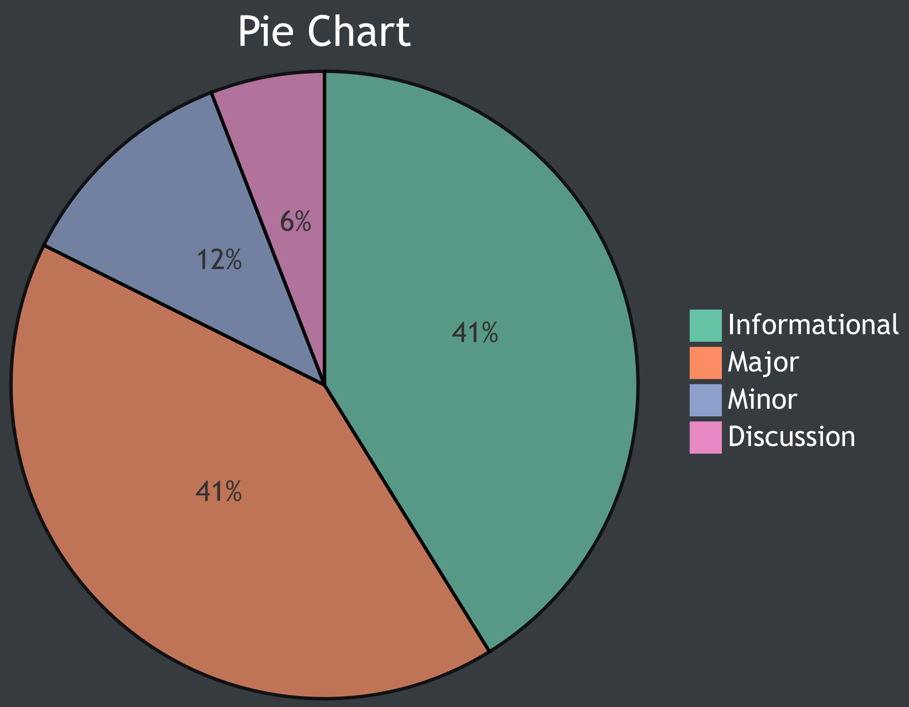
The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

File in Scope

ID	Contract	SHA256-Checksum
HCF	**hedgeyCallsFactoryV2.sol**	f4de0f4a988ada4902607789b879ba5814e8e20e70d6334993b225880b242d08
HC	**hedgeyCallsV2.sol**	d3e495e6d71e70a5df7a999e983b1619e9b0d9500de9211c25e059467c211e9e
HPF	**hedgeyPutsFactoryV2.sol**	8d9536e8322582ad8dbd415f745674eee5544f2910fabaa16d707f8e0eff3461
HP	**hedgeyPutsV2.sol**	da4c24103e4e655b3025453ea69975e2e77ab8af42301b180932f76cec052413
ML	**miscLib.sol**	7abc96f66d9dd8cfd042b2ad37288af2737f27c955195ff1af87a6163ac61571

Findings



ID	Title	Type	Severity	Resolved
HCF-01	Unrestricted Loops	Language Specific	🟢Informational	✓
HC-01	Unreturned Change	Logical Issue	🔵Minor	✓
HC-02	Undistinguishable Calls	Volatile Code	🟠 Major	✓
HC-03	Undistinguishable Calls	Volatile Code	🟠 Major	✓
HC-04	Big Loops	Language Specific	🟢Informational	✓
HC-05	Wrong Strict Equality	Logical Issue	🟠Major	✓
HPF-01	Big Loops	Language Specific	🟢Informational	✓
HP-01	Big Loops	Language Specific	🟢Informational	✓
HP-02	Unreturned Change	Logical Issue	🔵Minor	✓
HP-03	Undistinguishable Calls	Volatile Code	🟠Major	✓
HP-04	Undistinguishable Calls	Volatile Code	🟠Major	✓
HP-05	Unauthorized Purchase	Volatile Code	🟠Major	✓
HP-06	Wrong Library	Library Typo	🟢Informational	✓
HP-07	Network Configuration	Discussion	⚪Discussion	✓
HP-08	Wrong Strict Equality	Logical Issue	🟠Major	✓
HP-09	Redundant Code	Logical Issue	🟢Informational	✓
HP-10	Replace Declaration	Gas Optimization	🟢Informational	⏱

HCF-01: Unrestricted Loops

Type	Severity	Location
Language Specific	●Informational	hedgeyCallsFactoryV1.sol L42

Description:

Data view queries via full-nodes or API vendors may fail if the length of array is too large.

```
function getTotalContracts() public view returns (address[] memory contracts) {
        // The argument l will increase continuously
        uint l = totalContracts.length;
        contracts = new address[](l);
        for (uint i; i < l; i++) {
            contracts[i] = totalContracts[i];
        }
    }
```

Recommendation:

Limit the length of the returned array. For example, return only the last ten addresses. Or, given a range of the array, the view call will return the addresses specified. An example is below:

```
function getTotalContracts(uint256 _fromIndex, uint256 _toIndex) public view returns (address[] memory contracts) {
        uint256 l = totalContracts.length;
        require(_fromIndex <= _toIndex, "Illegal query length");
        require(_toIndex < l, "Overflow query length");
        uint256 length = _toIndex - _fromIndex + 1;
        contracts = new address[](length);
        for (uint256 i = _fromIndex; i <= _toIndex; i++) {
            contracts[i] = totalContracts[i];
        }
    }
```

Alleviation:

The development team decided to take an off-chain approach to solve this problem. They have removed these code in commit

73eda85698b6e03e9842be03b36bf640c84a7f7f

HC-01: Unreturned Change

Type	Severity	Location
Logical Issue	●Minor	hedgeyCallsV1.sol L173, L271

Description:

When a user sends more ethers than required, the change wouldn't be returned and the user would loses them.

In the `newBid()` function, when `pymtWeth` is `true`, the function of `newBid()` does not return changes:

```
function newBid(uint _assetAmt, uint _strike, uint _price, uint _expiry) payable public {
       ......
       uint balCheck = pymtWeth ? msg.value : IERC20(pymtCurrency).balanceOf(msg.sender);
       require(balCheck >= _price, "not enough cash to bid");
       depositPymt(pymtWeth, pymtCurrency, msg.sender, _price);
       ......
    }
```

A similar issue is found in the function of `newAsk()`:

```
function newAsk(uint _assetAmt, uint _strike, uint _price, uint _expiry) payable public {

      ......
      uint balCheck = assetWeth ? msg.value : IERC20(asset).balanceOf(msg.sender);
      require(balCheck >= _assetAmt, "not enough to sell this call option");
      depositPymt(assetWeth, asset, msg.sender, _assetAmt);
      ......
    }
```

Recommendation:

One recommended method is to return the change of ethers to the user directly. Or, use a variable to keep the amount of change, and let the user withdraw it.

Alleviation:

The developer team solved this problem by limiting the amount of ether in commit

3d8f96732ccdce28c0f5ab7c09bf63ab4b123178

HC-02: Undistinguishable Calls

Type	Severity	Location
Volatile Code	● Major	hedgeyCallsV1.sol L222

Description:

This contract does not separate a `call` generated by `newBid()` from ones by `newAsk()`, leading to severe vulnerability. In the `sellOpenOptionToNewBid()` function, the local variable `newBid` can point to a `call` (from `calls[_d]`) generated by `newAsk()`, thus the funds are transferred incorrectly:

```
function sellOpenOptionToNewBid(uint _c, uint _d) payable public nonReentrant {
        ......
        //The argument called[ _d] can be generated by the NewAsk() function.
        Call storage newBid = calls[_d];
        require(msg.sender == openCall.long, "you dont own this");
        require(openCall.strike == newBid.strike, "not the right strike");
        require(openCall.assetAmt == newBid.assetAmt, "not the right assetAmt");
        require(openCall.expiry == newBid.expiry, "not the right expiry");
        require(openCall.open && !newBid.open && newBid.tradeable && !openCall.exercised && !newBid.exercised && openCall.expiry > now && newBid.expiry > now, "something is wrong");
        newBid.exercised = true;
        newBid.tradeable = false;
       ......
    }
```

Recommendation:

One recommended method is to add a flag to the `call` struct to show if a `call` is of `newBid` or `newAsk`.

Alleviation:

The developer team achieved the same goal by distinguishing whether the value of the parameter `newBid.short` is `0x0` or not in commit

b7f47e7f56946cd00b2dcb6ca4cc3d9230438168

HC-03: Undistinguishable Calls

Type	Severity	Location
Volatile Code	● Major	hedgeyCallsV1.sol L346

Description:

Same issue as HC-02, in the `buyOptionFromNewShort()` function, the `call` loaded into local variable `newAsk` could be a `call` from `newBid()` , therefore the funds are transferred incorrectly:

```
 function buyOptionFromNewShort(uint _c, uint _d) payable public nonReentrant {
        ......
        //The argument called[ _d] can be generated by the NewBid() function.
        Call storage newAsk = calls[_d];

        require(msg.sender == openShort.short, "your not the short");
        require(openShort.strike == newAsk.strike, "not the right strike");
        require(openShort.assetAmt == newAsk.assetAmt, "not the right assetAmt");
        require(openShort.expiry == newAsk.expiry, "not the right expiry");
        require(openShort.open && !newAsk.open && newAsk.tradeable && !openShort.exercised && !newAsk.exercised && openShort.expiry > now && newAsk.expiry > now, "something is wrong");
        newAsk.exercised = true;
        newAsk.tradeable = false;
        newAsk.open = false;
        ......
    }
```

Recommendation:

One recommended method is to add a flag to the `call` struct to show if a `call` is of `newBid` or `newAsk` .

Alleviation:

The developer team achieved the same goal by distinguishing between the equality of the parameter `newAsk.short` and `newAsk.long` in commit

b7f47e7f56946cd00b2dcb6ca4cc3d9230438168

HC-04: Big Loops

Type	Severity	Location
Language Specific	●Informational	hedgeyCallsV1.sol L124,L133,L142,L150,L160

Description:

It's wrong to return an entire unbounded array, because the RPC-calls might fail. As the value of `c` increasing, the queries performed by these functions wouldn't fetch the expected results.

```
function getOpenOptions() public view returns (int256[] memory _calls) {
    _calls = new int256[](c);
    for (uint i = 0; i < c; i++) {
        ......
     }
   }


//gets all of the calls that someone is short
function getShortOptions(address _short) public view returns (int[] memory _calls) {
        _calls = new int[](c);
        for (uint i = 0; i < c; i++) {
            ......
        }
    }

function getLongOptions(address _long) public view returns (int[] memory _calls) {
        _calls = new int[](c);
        for (uint i = 0; i < c; i++) {
            ......
        }
    }

function getAllOptions(address _holder) public view returns (int[] memory _calls) {
        _calls = new int[](c);
        for (uint i = 0; i < c; i++) {
            ......
        }
    }


function getTradeableOptions() public view returns (int[] memory _calls) {
        _calls = new int[](c);
        for (uint i = 0; i < c; i++) {
            ......
        }
    }
```

Recommendation:

Give a range of the array. For example, give [0,10) for returning first ten addresses. A recommended method is shown in HCF-01.

Alleviation:

The development team has removed these codes in commit

e4d9eee2da3efede6c9acca4278f800b0a0c58c2

HC-05: Wrong Strict Equality

Type	Severity	Location
Logical Issue	●Major	hedgeyCallsV1.sol L96

Description:

The function `transferPymt()` is called twice in `buyNewOption()` , `buyOptionFromNewShort()` , `buyOpenOption()` . The checking `msg.value == _amt` in `` `transferPymt() `` will cause these three functions reverted.

```
    function transferPymt(bool _isWETH, address _token, address from, address payable to, uint _amt) internal {
        if (_isWETH) {
            if (!Address.isContract(to)) {
                require(msg.value == _amt, "transfer issue: transferring wrong eth amount");
                to.transfer(_amt);
            } else {
                // we want to deliver WETH from ETH here for better handling at contract
                IWETH(weth).deposit{value: _amt}();
                assert(IWETH(weth).transfer(to, _amt));
            }
        } else {
            SafeERC20.safeTransferFrom(IERC20(_token), from, to, _amt);
        }
    }
```

Alleviation:

The development team has solved this problem by using `transferPymtWithFee()` function instead of `transferPymt()` function in commit

78b7fab3dd2d60d15d243dee388935fd2caf661e

HPF-01: Big Loops

Type	Severity	Location
Language Specific	●Informational	hedgeyPutsFactoryV1.sol L42

Description:

It's wrong to return an entire unbounded array, because the RPC-calls might fail. As the value of `c` increasing, the queries performed by these functions wouldn't fetch the expected results.

```
function getTotalContracts() public view returns (address[] memory contracts) {
        // The argument l will increase continuously
        uint l = totalContracts.length;
        contracts = new address[](l);
        for (uint i; i < l; i++) {
            contracts[i] = totalContracts[i];
        }
    }
```

Recommendation:

Give a range of the array. For example, give [0,10) for returning first ten addresses. A recommended method is shown in HCF-01.

Alleviation:

The developer team has removed this part of code in commit

dced7f81cfb0a701f61fdc7b7bccf84d3a7c8e9f

HP-01: Big Loops

Type	Severity	Location
Language Specific	●Informational	hedgeyPutsV1.sol L122,L133,L142,L150,L159

Description:

It's wrong to return an entire unbounded array, because the RPC-calls might fail. As the value of `p` increasing, the queries performed by these functions wouldn't fetch the expected results.

```
function getOpenOptions() public view returns (int[] memory _puts) {
       _puts = new int[](p);
       for (uint i = 0; i < p; i++) {
           ......
       }
   }


   //put call getters
   function getShortOptions(address _short) public view returns (int[] memory _puts) {
       _puts = new int[](p);
       for (uint i = 0; i < p; i++) {
           .......
       }
   }


   function getLongOptions(address _long) public view returns (int[] memory _puts) {
       _puts = new int[](p);
       for (uint i = 0; i < p; i++) {
           ......
       }
   }


   function getAllOptions(address _holder) public view returns (int[] memory _puts) {
       _puts = new int[](p);
       for (uint i = 0; i < p; i++) {
           ......
       }
   }


   function getTradeableOptions() public view returns (int[] memory _puts) {
       _puts = new int[](p);
       for (uint i = 0; i < p; i++) {
           ......
       }
   }
```

Recommendation:

Give a range of the array to the functions. For example, give [0,10) for returning first ten addresses. A recommended method is shown in HCF-01.

Alleviation:

The developer team has removed this part of code in commit

7fbcea92d6255d859dda3111e71457a5c419cbbb

HP-02: Unreturned Change

Type	Severity	Location
Logical Issue	🔵Minor	hedgeyPutsV1.sol L171,L272

Description:

When a user sends more ethers than required, the change wouldn't be returned.

In the `newBid()` function, when `pymtWeth` is `true` , the function of `newBid()` does not return change:

```
function newBid(uint _assetAmt, uint _strike, uint _price, uint _expiry) payable public {
        ......
        uint balCheck = pymtWeth ? msg.value : IERC20(pymtCurrency).balanceOf(msg.sender);
        require(balCheck >= _price, "insufficent purchase cash");
        depositPymt(pymtWeth, pymtCurrency, msg.sender, _price); //handles weth and token deposits into contract
        ......
    }
```

A similar issue is found in the function of `newAsk()` :

```
 function newAsk(uint _assetAmt, uint _strike, uint _price, uint _expiry) payable public {
        ......
        uint balCheck = pymtWeth ? msg.value : IERC20(pymtCurrency).balanceOf(msg.sender);
        require(balCheck >= _totalPurch, "you dont have enough collateral to write this option");
        depositPymt(pymtWeth, pymtCurrency, msg.sender, _totalPurch);
        ......
    }
```

Recommendation:

One recommended method is to return the change of ethers to the user directly. Or, use a variable to keep the amount of change, and let the user withdraw it.

Alleviation:

The development team heeded our advice and resolved this issue in commit

00c4190fe991553d3d422e3ab380e085a64d7173

HP-03: Undistinguishable Calls

Type	Severity	Location
Volatile Code	● Major	hedgeyPutsV1.sol L222

Description:

This contract does not separate the structures generated by `newBid()` from ones by `newAsk()` , leading to severe results. In the `sellOpenOptionToNewBid()` function, the local variable `newBid` can point to a `put` (from `puts[_q]`) generated by `newAsk()` , thus the funds are transferred incorrectly:

```
function sellOpenOptionToNewBid(uint _p, uint _q) payable public nonReentrant {
        ......
        //The argument puts[_q] can be generated by the NewAsk() function.
        Put storage newBid = puts[_q];
        require(msg.sender == openPut.long, "you dont own this");
        require(openPut.strike == newBid.strike, "not the right strike");
        require(openPut.assetAmt == newBid.assetAmt, "not the right assetAmt");
        require(openPut.expiry == newBid.expiry, "not the right expiry");
        require(openPut.open && !newBid.open && newBid.tradeable && !openPut.exercised && !newBid.exercised && openPut.expiry > now && newBid.expiry > now, "something is wrong");
        //close out our new bid
        newBid.exercised = true;
        newBid.tradeable = false;
        ......
    }
```

Recommendation:

One recommended method is to add a flag to the `put` struct to show if a `put` is of `newBid()` or `newAsk()` .

Alleviation:

The developer team achieved the same goal by distinguishing whether the value of the parameter `newBid.short` is `0x0` or not in commit

eebc9261a5885719c2f8edb6ce907d2c3b0846fa

HP-04: Undistinguishable Calls

Type	Severity	Location
Volatile Code	●Major	hedgeyPutsV1.sol L348

Description:

Same issue as HP-03, in the `buyOptionFromNewShort()` function, the `put` loaded into local variable `newAsk()` could be a `put` from `newBid()` , therefore the funds are transferred incorrectly:

```
function buyOptionFromNewShort(uint _p, uint _q) payable public nonReentrant {
        ......
        //The argument puts[_q] can be generated by the NewBid() function.
        Put storage newAsk = puts[_q];
        //everything needs to match
        require(msg.sender == openShort.short, "your not the short");
        require(openShort.strike == newAsk.strike, "not the right strike");
        require(openShort.assetAmt == newAsk.assetAmt, "not the right assetAmt");
        require(openShort.expiry == newAsk.expiry, "not the right expiry");
        require(openShort.open && !newAsk.open && newAsk.tradeable && !openShort.exercised &&
!newAsk.exercised && openShort.expiry > now && newAsk.expiry > now, "something is wrong");
        newAsk.exercised = true;
        newAsk.tradeable = false;
        newAsk.open = false;
        ......
    }
```

Recommendation:

One recommended method is to add a flag to the `put` struct to show if a `put` is of `newBid()` or `newAsk()` .

Alleviation:

The developer team achieved the same goal by distinguishing between the equality of the parameter `newAsk.short` and `newAsk.long` in commit

7fbcea92d6255d859dda3111e71457a5c419cbbb

HP-05: Unauthorized Purchase

Type	Severity	Location
Logical Issue	●Major	hedgeyPutsV1.sol L386

Description:

The flag `tradeable` is set to indicate whether the option owner is willing to sell, the absence of this flag checking could allow a malicious user to be able to modify someone else's option without permission.

```
function buyOpenOption(uint _p) payable public nonReentrant {
        Put storage put = puts[_p];
        require(msg.sender != put.long, "You already own this");
        require(put.open, "This call isnt opened yet");
        require(put.expiry >= now, "This call is already expired");
        require(!put.exercised, "This has already been exercised!");
        ......
    }
```

Recommendation:

One recommended approach is to add a `put.tradeable` checking that would avoid the above issue:

```
function buyOpenOption(uint _p) payable public nonReentrant {
        Put storage put = puts[_p];
        require(msg.sender != put.long, "You already own this");
        require(put.open, "This call isnt opened yet");
        require(put.expiry >= now, "This call is already expired");
        require(!put.exercised, "This has already been exercised!");
        require(put.tradeable, "not tradeable");
        ......
    }
```

Alleviation:

The development team heeded our advice and resolved this issue in commit

5df7976aec3b36af9b9bcb3f1696ce47325eef83

HP-06: Wrong Library

Type	Severity	Location
Library Typo	●Informational	hedgeyPutsV1.sol L3

Description:

Spelling error of the name of the imported library:

```
import './misLib.sol';
```

Recommendation:

```
import './miscLib.sol';
```

Alleviation:

The development team heeded our advice and resolved this issue in commit

61e17fd28c8adfcb0b3a19fe6e8f0e34b56f68a8

HP-07: Network Configuration

Type	Severity	Location
Discussion	● Discussion	hedgeyPutsV1.sol L21,L26 hedgeyCallsV1.sol L20,L25

Description:

Note that both of these two contract addresses are on the kovan testnet, not the addresses on the mainnet. Please don't forget to set them correctly before the deployment on the mainnet.

```
// In contract of HedgeyPutsV1
contract HedgeyPutsV1 is ReentrancyGuard {
   ......
   address payable public weth = 0xd0A1E359811322d97991E03f863a0C30C2cF029C;
   ......
   address public uniFactory = 0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f;
   ......
}
```

Alleviation:

The issue has been discussed.

HP-08: Wrong Strict Equality

Type	Severity	Location
Logical Issue	🟠Major	hedgeyPutsV1.sol L94

Description:

The same issue as HC-05.

Alleviation:

The development team has solved this problem by using `transferPymtWithFee()` function instead of `transferPymt()` function in commit

e3674f1158c5d24ccb51a42f0f46e1fd9dbfd026

HP-09: Redundant Code

Type	Severity	Location
Logical Issue	●Informational	hedgeyPutsV1.sol L294

Description:

The condition `require(put.tradable, "this is not a tradable option");` has confirmed that the value of `put.tradable` is `true`, so there is no need to repeat the assignment.

```
function changeNewOption(uint _p, uint _assetAmt, uint _strike, uint _price, uint _expiry) payable public nonReentrant {
        ......
        require(put.tradeable, "this is not a tradeable option");
        ......
        if (msg.sender == put.short) {
            ......
            put.tradeable = true;
            ......
        } else if (put.short == address(0x0)) {
            ......
            put.tradeable = true;
            ......
    }
```

The same issue appears in `hedgeyCallsV1.sol`.

Alleviation:

The issue has been discussed. The development team has not modified this issue in current code.

HP-10: Replace Declaration

Type	Severity	Location
Logical Issue	●Informational	hedgeyPutsV2.sol L144

Description:

The declaration of `public` functions that are never called by the contract should be declared `external` to save gas.

For example, some functions are as follows:

```
function updateAMM() public {
        ......
    }

function newBid(uint _assetAmt, uint _strike, uint _price, uint _expiry) payable public {
        ......
    }

function cancelNewBid(uint _p) public nonReentrant {
        ......
    }

 function sellOpenOptionToNewBid(uint _p, uint _q, uint _price) payable public nonReentrant {
        ......
    }
......
```

The same issue exists in `hedgeyCallsV2.sol` contract.

Recommendation:

Use the `external` attribute for functions never called from the contract.

Alleviation:

The issue has been discussed.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete` .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Icons explanation

✓ : Issue resolved

❗ : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

❗✓ : Issue partially resolved. Not all instances of an issue was resolved.