SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Squeeth

Prepared for:	Opyn (Crab V2)
Prepared by:	Sherlock
Lead Security Expert:	shw
Security Experts:	DefSec, rayn
Dates Audited:	June 29th - July 4th, 2022
Prepared on:	July 5th, 2022

Introduction

The Squeeth contract is designed for users to long or short a special index: Eth^2, as an implementation of a Power Perpetual.

The Crab Strategy contract handles deposits and withdrawals, and automatically modifies its mixture of Squeeth debt and ETH collateral to create a position with an approximate delta of 0 to the price of ETH (neutral exposure to ETH).

This audit focused on V2 of the Crab Strategy contract.

Some of the features of Crab V2 include: a new hedging mechanism, ETH or USDC deposits, partial fills of hedges, and variable hedging frequency.

Scope

Branch: main
Commit: ea61adfadaf57a2a19dc1d6a78729bebea1f967c
Contracts:

- /strategy/CrabHelper.sol
- /strategy/CrabMigration.sol
- /strategy/CrabStrategyV2.sol
- /strategy/helper/StrategySwap.sol

Protocol Attributes

Language: Solidity
Assembly usage: No
Solc version: 0.7.6

Blockchain: Ethereum
L2s: None

Findings

Each issue has an assigned severity:

- Informational issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgement as to whether to address such issues.
- Low issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Total Issues

Informational	Low	Medium	High
18	9	1	0

Issue M-1 `hedgingTwapPeriod` can exceed pool maximum twap period

Summary

`hedgingTwapPeriod` can exceed pool maximum twap period.

Severity

Medium

Vulnerability Detail

On the `Oracle` contract's `getTwap` function, `_period` means that number of seconds in the past to start calculating time-weighted average. In the `CrabStrategyV2` contract, the period is set with `hedgingTwapPeriod` with `_checkPeriod: true`. However, if `hedgingTwapPeriod` exceeds the pool's maximum twap period, that period is used, but `hedgingTwapPeriod` is ignored.

Impact

Unexpected twap price calculation without `hedgingTwapPeriod`.

Code Snippet

CrabStrategyV2.solL786,Oracle.solL45

```
function _isPriceHedge() internal view returns (bool) {
    uint256 wSqueethEthPrice = IOracle(oracle).getTwap(ethWSqueethPool,
↪   wPowerPerp, weth, hedgingTwapPeriod, true);
    uint256 cachedRatio = wSqueethEthPrice.wdiv(priceAtLastHedge);
    uint256 priceThreshold = cachedRatio > 1e18 ? (cachedRatio).sub(1e18) :
↪   uint256(1e18).sub(cachedRatio);

    return priceThreshold >= hedgePriceThreshold;
}
```

Tool used

Manual Review

Recommendation

In the `setHedgingTwapPeriod()`, consider adding a check to ensure the pool's maximum twap period is at least `hedgingTwapPeriod` to let the set action take effect right after the transaction execution and avoid unexpected results as well.

Team

Opyn Team states that the manager would be responsible for increasing the storage slots on the uniswap oracle before changing this value. Opyn Team is waiting to change the `hedgingTwapPeriod` after increasing the storage slots to have the storage slots fill up does not result in a different behavior vs increasing it right after an increase in storage slots for the uniswap oracle. We don't anticipate this is something that will be changed frequently, if at all.

Sherlock

The issue marked as an acknowledged.

Issue L-1 Potential rounding error issues

Summary

Potential rounding error issues in `CrabMigration` can cause the last user to be unable to claim strategy tokens.

Severity

Low

Vulnerability Detail

The `claimV2Shares()` function of `CrabMigration` calculates the pro-rata `crabV2` shares based on users' deposited `crabV1` amount. The formula uses `wmul` and `wdiv`, which round up the calculation results. Therefore, the result, `amountV2ToTransfer`, can be larger than the ideal amount the user is supposed to get, while the difference is extremely small.

Impact

Getting more (but only dust) strategy tokens isn't an issue in most cases. However, the last user calling this function can fail to claim all their strategy tokens because of a lack of dust in the contract.

Code Snippet

CrabMigration.solL276-L281

Tool used

Manual Review

Recommendation

Alternatively, the last user can call `claimAndWithdraw` to claim their strategy tokens to avoid the lock of funds. Applying a round-down calculation instead is also a possible solution. In the long term, carefully examine the use of round-up calculations to prevent issues caused by token dust.

Team

Fixed in https://github.com/opynfinance/squeeth-monorepo/pull/562.

Sherlock

Verified.

Issue L-2 Missing validation on `dToken`

Summary

The `dToken` variable in `CrabMigration` lacks a validation check.

Severity

Low

Vulnerability Detail

The `dToken` provided through the construction of `CrabMigration` is not validated to have the underlying token as `WETH`, and therefore there is no strong connection between the two variables.

Impact

Providing the wrong `dToken` parameter would cause critical functions such as `onDeferredLiquidityCheck()` to be broken. Crab strategy tokens would not be able to migrate successfully.

Code Snippet

CrabMigration.solL121

Tool used

Manual Review

Recommendation

Add a check `_dToken.underlyingAsset()==_weth` in the constructor.

Team

Implemented in https://github.com/opynfinance/squeeth-monorepo/pull/582.

Sherlock

Verified.

Issue L-3 Approve function return value is ignored

Summary

Erc20 approve function return value is ignored in the code base. If the tokens don't correctly implement the EIP20 standard and the approve function returns void instead of a success boolean.

Severity

Low

Vulnerability Detail

Tokens that are not compliant with the `ERC20` specification could return false from the `approve` function call to indicate the approval fails, while the calling contract would not notice the failure if the return value is not checked.

Impact

Approve function will fail but will not revert when the tokens are not compliant with the `ERC20` specification.

Code Snippet

StrategySwap.solL43

```
function _swapExactInputSingle(
    address _tokenIn,
    address _tokenOut,
    address _from,
    address _to,
    uint256 _amountIn,
    uint256 _minAmountOut,
    uint24 _fee
) internal returns (uint256 amountOut) {
    // _from must approve this contract

    // Transfer the specified amount of tokenIn to this contract.
    IERC20(_tokenIn).transferFrom(_from, address(this), _amountIn);

    // Approve the router to spend tokenIn.
    IERC20(_tokenIn).approve(address(swapRouter), _amountIn);

    // We also set the sqrtPriceLimitx96 to be 0 to ensure we swap our exact
↪   input amount.
```

```
    ISwapRouter.ExactInputSingleParams memory params =
 ↪  ISwapRouter.ExactInputSingleParams({
        tokenIn: _tokenIn,
        tokenOut: _tokenOut,
        fee: _fee,
        recipient: _to,
        deadline: block.timestamp,
        amountIn: _amountIn,
        amountOutMinimum: _minAmountOut,
        sqrtPriceLimitX96: 0
    });

    // The call to `exactInputSingle` executes the swap.
    amountOut = swapRouter.exactInputSingle(params);
}
```

Tool used

Manual Review

Recommendation

Consider using the `safeApprove` function instead of `approve`, which reverts the transaction with a proper error message when the return value of approval is false. On the other hand, a better approach is to use the `safeIncreaseAllowance` function, which mitigates the race condition on ERC20 tokens.

Team

We only plan to support conforming tokens at this time. As this is a helper contract, we can upgrade at a later date if needed.

Sherlock

Acknowledged.

Issue L-4 Tokens with fee on transfer not supported

Summary

The `StrategySwap.sol` does not support fee on transfer tokens.

Severity

Low

Vulnerability Detail

The code base does not support `rebasing/deflationary/inflationary` tokens whose balance changes during transfers or over time. If the transferred token is a transfer-on-fee/deflationary token, the actual received amount could be less than `_amountIn`.

Impact

The `exactInputSingle()` will revert if the token gets fee during the transfer. Swap operation cannot be completed with amount after fee.

Code Snippet

StrategySwap.solL40

```
function _swapExactInputSingle(
    address _tokenIn,
    address _tokenOut,
    address _from,
    address _to,
    uint256 _amountIn,
    uint256 _minAmountOut,
    uint24 _fee
) internal returns (uint256 amountOut) {
    // _from must approve this contract

    // Transfer the specified amount of tokenIn to this contract.
    IERC20(_tokenIn).transferFrom(_from, address(this), _amountIn);

    // Approve the router to spend tokenIn.
    IERC20(_tokenIn).approve(address(swapRouter), _amountIn);

    // We also set the sqrtPriceLimitx96 to be 0 to ensure we swap our exact
↪   input amount.
    ISwapRouter.ExactInputSingleParams memory params =
↪   ISwapRouter.ExactInputSingleParams({
```

SHERLOCK

```
            tokenIn: _tokenIn,
            tokenOut: _tokenOut,
            fee: _fee,
            recipient: _to,
            deadline: block.timestamp,
            amountIn: _amountIn,
            amountOutMinimum: _minAmountOut,
            sqrtPriceLimitX96: 0
        });

        // The call to `exactInputSingle` executes the swap.
        amountOut = swapRouter.exactInputSingle(params);
    }
```

Tool used

Manual Review

Recommendation

Determine the transferred amount by subtracting the before & after balance.

Team

We only plan to support non-fee on transfer tokens at this time. As this is a helper contract, we can upgrade at a later date if needed.

Sherlock

Acknowledged.

Issue L-5 `crabV2` initialization can revert if the `strategyCap` is not set

Summary

`crabV2` initialization can revert if the `strategyCap` is not set on the migration contract.

Severity

Low

Vulnerability Detail

During the initialization of the `crabV2` contract, the strategy cap is checked through the `_checkStrategyCap` function. If the owner does not set strategyCap with `setStrategyCap` function, the `initialize` function will revert with pre-defined `msg.value`. The `strategyCap` default is set to `0`.

Impact

`_flashCallback` function will revert during the `crabV2` initialization.

Code Snippet

CrabStrategyV2.solL211

```
function initialize(
    uint256 _wSqueethToMint,
    uint256 _crabSharesToMint,
    uint256 _timeAtLastHedge,
    uint256 _priceAtLastHedge
) external payable {
    require(msg.sender == crabMigration, "not Crab Migration contract");
    require(!isInitialized, "Crab V2 already initialized");

    uint256 amount = msg.value;
    uint256 strategyDebt;
    uint256 strategyCollateral;

    _checkStrategyCap(amount, strategyCollateral);

    require((strategyDebt == 0 && strategyCollateral == 0), "C5");
    // store hedge data from crab v1
    timeAtLastHedge = _timeAtLastHedge;
    priceAtLastHedge = _priceAtLastHedge;
    // mint wSqueeth and send it to msg.sender
    _mintWPowerPerp(msg.sender, _wSqueethToMint, amount, false);
    // mint LP to depositor
    _mintStrategyToken(msg.sender, _crabSharesToMint);

    isInitialized = true;
}
```

Tool used

Manual Review

Recommendation

Ensure that `strategyCap` is set before initialization.

Team

We aren't fixing this directly, but this has been indirectly resolved by requiring the manager specifies the strategy cap when calling initialize() and not letting it being set via the setStrategyCap until after initialization. PR: https://github.com/opynfinance/squeeth-monorepo/pull/573

Sherlock

Verified.

Issue L-6 Future deposits can still be allowed after transferring vault

Summary

Future deposits can still be allowed after transferring vault to new strategy contract.

Severity

Low

Vulnerability Detail

`transferVault` function has been used to transfer the vault to the new strategy contract. In the function, `strategyCap` has been set to `0` for preventing future deposits. But, still owner can change the strategy cap through `setStrategyCap` function to allow future deposits.

Impact

Although the vault is transferred to a new strategy contract, the future deposit can be enabled by an owner.

Code Snippet

CrabStrategyV2.solL243

```
function setStrategyCap(uint256 _capAmount) external onlyOwner {
    _setStrategyCap(_capAmount);
}
```

Tool used

Manual Review

Recommendation

Ensure that future deposits are not allowed with `setStrategyCap` function.

Team

Opyn Team states that If the manager set the cap above 0 post transfer, any deposit or flashDeposit call would still revert as it would try to mint oSQTH from a vault it doesn't own anymore.

Sherlock

Acknowledged.

Issue L-7 Use `safeTransfer/safeTransferFrom` consistently instead of `transfer/transferFrom`

Summary

Replace `transferFrom()` with `safeTransferFrom()` since `_tokenIn` can be any `ERC20` token implementation. If `transferFrom()` does not return a value (e.g., USDT), the transaction reverts because of a decoding error.

Severity

Informational

Vulnerability Detail

It is good to add a require() statement that checks the return value of token transfers or to use something like OpenZeppelin's safeTransfer/safeTransferFrom unless one is sure the given token reverts in case of a failure. Failure to do so will cause silent failures of transfers and affect token accounting in the contract.

In the current code base, only CrabHelper.sol interacts with the _swapExactInputSingle function. Although, the current implementation does not pose any risk, the future implementation should consider using `safeTransfer/safeTransferFrom`.

Impact

Revert without error.

Code Snippet

StrategySwap.solL28

```
function _swapExactInputSingle(
    address _tokenIn,
    address _tokenOut,
    address _from,
    address _to,
    uint256 _amountIn,
    uint256 _minAmountOut,
    uint24 _fee
) internal returns (uint256 amountOut) {
    // _from must approve this contract

    // Transfer the specified amount of tokenIn to this contract.
    IERC20(_tokenIn).transferFrom(_from, address(this), _amountIn);
    // Approve the router to spend tokenIn.
    IERC20(_tokenIn).approve(address(swapRouter), _amountIn);

    // We also set the sqrtPriceLimitx96 to be 0 to ensure we swap our exact
 ↪  input amount.
    ISwapRouter.ExactInputSingleParams memory params =
 ↪  ISwapRouter.ExactInputSingleParams({
        tokenIn: _tokenIn,
        tokenOut: _tokenOut,
        fee: _fee,
        recipient: _to,
        deadline: block.timestamp,
        amountIn: _amountIn,
        amountOutMinimum: _minAmountOut,
        sqrtPriceLimitX96: 0
    });

    // The call to `exactInputSingle` executes the swap.
    amountOut = swapRouter.exactInputSingle(params);
}
```

Tool used

Manual Review

Recommendation

Consider using safeTransfer/safeTransferFrom or require() consistently.

Team

We agree that some tokens could fail to successfully swap, but we only plan to support conforming tokens at this time. This is a helper contract, so we can upgrade at a later date if we need to support a non-conforming token. We don't see any risk other than limited functionality (reverts) for some tokens.

Sherlock

Acknowledged.

Issue L-8 User `crabV1` tokens can get stuck when the owner does not launch migrate

Summary

User cannot withdraw their assets when the owner does not launch migration.

Severity

Low

Vulnerability Detail

Users can deposit crab v1 shares in the pool through `depositV1Shares` function. However, when the owner has not launched the migration, the user is not able to withdraw `crabV1` tokens from the contract.

Impact

User tokens can get stuck in the contract.

Code Snippet

CrabMigration.solL138

```
function depositV1Shares(uint256 amount) external afterInitialized
↪   beforeMigration {
    sharesDeposited[msg.sender] += amount;
    totalCrabV1SharesMigrated += amount;
    crabV1.transferFrom(msg.sender, address(this), amount);
}
```

Tool used

Manual Review

Recommendation

Consider adding ability to withdraw assets before migration.

Team

Fixed with https://github.com/opynfinance/squeeth-monorepo/pull/585/files.

Sherlock

Verified.

Issue L-9 Missing validation in the constructor

Summary

There is no zero address check for contract constructor.

Severity

Low

Vulnerability Detail

The contract constructor is missing validation mechanisms.

Impact

If the `EULER_MAINNET` is assigned to zero address, the change will break the protocol functionality.

Code Snippet

CrabMigration.solL119

```
constructor(
    address payable _crabV1,
    address _weth,
    address _eulerExec,
    address _dToken,
    address _eulerMainnet
) {
    crabV1 = CrabStrategy(_crabV1);
    euler = IEulerExec(_eulerExec);
    EULER_MAINNET = _eulerMainnet;
    weth = WETH9(_weth);
    dToken = _dToken;
    wPowerPerp = crabV1.wPowerPerp();
}
```

Tool used

Manual Review

Recommendation

Add zero address checks in the constructor for the state variables.

Team

Fixed in https://github.com/opynfinance/squeeth-monorepo/pull/572.

Sherlock

Verified.

Issue L-10 Missing two-step transfer ownership pattern

Summary

The Opyn contracts use `Ownable` from `OpenZeppelin` which is a simple mechanism to transfer the ownership not supporting a two-step transfer ownership pattern.

Severity

Low

Vulnerability Detail

`Ownable` is a simpler mechanism with a single owner "role" that can be assigned to a single account. This simpler mechanism can be useful for quick tests but projects with production concerns are likely to outgrow it.

Impact

Transferring ownership is a critical operation and this could lead to transferring it to an inaccessible wallet or renouncing the ownership e.g. by mistake.

Tool used

Manual Review

Recommendation

It is recommended to implement a two-step transfer ownership mechanism where the ownership is transferred and later claimed by a new owner to confirm the whole process and prevent lockout. Note: this is relevant for all the contracts `CrabStrategyV2.sol`, `CrabMigration.sol`.

As OpenZeppelin ecosystem does not provide such implementation it has to be done in-house. For the inspiration `BoringOwnable` can be considered, however it has to be well tested, especially if it is integrated with other OpenZeppelin contracts used by the project.

References

- https://docs.openzeppelin.com/contracts/4.x/api/access
- https://github.com/boringcrypto/BoringSolidity/blob/master/contracts/BoringOwnable.sol

Team

We may consider this in the future for future strategy contracts, but would want to spend more time with in house development and testing.

Sherlock

Acknowledged.

Issue I-1 `flashWithdrawERC20` and `flashDepositERC20` functions are not working with `weth`

Summary

In the flash deposit and withdraw functions, `weth` can not be passed as `_tokenIn` and `_tokenOut` because `_swapExactInputSingle` already uses `weth` as `_tokenIn` or `_tokenOut`.

Severity

Informational

Vulnerability Detail

In the flash deposit and withdraw functions, weth cannot be passed as _tokenIn and _tokenOut because `_swapExactInputSingle` already uses `weth` as `_tokenIn` or `_tokenOut`.

Impact

`flashWithdrawERC20` and `flashDepositERC20` are not working with `weth` function parameter.

Code Snippet

CrabHelper.solL51, CrabHelper.solL74

```
    function flashDepositERC20(
        uint256 _ethToDeposit,
        uint256 _amountIn,
        uint256 _minEthToGet,
        uint24 _fee,
        address _tokenIn
    ) external nonReentrant {
        _swapExactInputSingle(_tokenIn, weth, msg.sender, address(this),
 ↪  _amountIn, _minEthToGet, _fee);
    ....
}

    function flashWithdrawERC20(
        uint256 _crabAmount,
        uint256 _maxEthToPay,
        address _tokenOut,
        uint256 _minAmountOut,
        uint24 _fee
    ) external nonReentrant {
...
        uint256 tokenReceived = _swapExactInputSingle(
            weth,
            _tokenOut,
            address(this),
            msg.sender,
            ethBalance,
            _minAmountOut,
            _fee
        );
...
}
```

Tool used

Manual Review

Recommendation

Add custom logic to handle flash deposit/withdraw with `WETH` if it is suitable with design. If it is not needed, it is recommended to add a check such as "`_tokenIn` or `_tokenOut` cannot be `weth`".

Team

Weth is hardcoded in those functions as it is required for the swap to work correctly. Weth to weth swaps wouldn't be useful in the system.

Sherlock

Acknowledged.

Issue I-2 Some error strings can be clarified

Summary

Some error strings can be clarified.

Severity

Informational

Vulnerability Detail

Some of the error strings in `CrabStrategyV2` can be rewritten to improve code clarity on error handling, for example:

- At L664, the error string "Time or Price is not within range" is unclear. As we understand it, the time or the price should exceed a threshold/range to trigger a hedge.
- At L674, the error string "Orders must be buying when hedge is selling" only describes one case (i.e., the hedge is selling) but not the other.

Impact

Precise error descriptions can improve code clarity.

Code Snippet

CrabStrategyV2.solL664, CrabStrategyV2.solL674

Tool used

Manual Review

Recommendation

Rewrite the error strings to be more precise.

Team

Implemented in https://github.com/opynfinance/squeeth-monorepo/pull/563.

Sherlock

Verified.

Issue I-3 Remove unnecessary code

Summary

In `CrabMigration`, `_flashCallback()` includes an unnecessary token approval to `crab V1`.

Severity

Informational

Vulnerability Detail

When users withdraw from `crabV1`, the `_burn` is called internally without invoking a token transfer. Therefore, users don't have to approve `crabV1` to transfer Crab strategy tokens from them before withdrawing from `crabV1`.

Impact

Save gas and improve code maintainability.

Code Snippet

CrabMigration.solL214

Tool used

Manual Review

Recommendation

Remove the unnecessary code.

Team

Implemented in https://github.com/opynfinance/squeeth-monorepo/pull/559.

Sherlock

Verified.

Issue I-4 Use `IWETH9` interface instead of `IERC20`

Summary

In the `CrabStrategyV2` contract, `weth` interaction is completed through `IERC20` interface. But, the `IWETH9` interface is already inherited from `IERC20`.

Severity

Informational

Vulnerability Detail

In the `CrabStrategyV2` contract, `weth` interaction is completed through `IERC20` interface. But, the `IWETH9` interface is already inherited from `IERC20`. For that reason, `weth` interactions can be completed with the `IWETH9` interface.

Impact

Improve consistency.

Code Snippet

CrabStrategyV2.solL630, CrabStrategyV2.solL638

```
        IERC20(weth).transferFrom(_order.trader, address(this), wethAmount);
        IWETH9(weth).withdraw(wethAmount);
        _mintWPowerPerp(_order.trader, _order.quantity, wethAmount, false);
    } else {
        // trader sends oSQTH and receives weth
        _burnWPowerPerp(_order.trader, _order.quantity, wethAmount, false);
        //wrap it
        IWETH9(weth).deposit{value: wethAmount}();
        IERC20(weth).transfer(_order.trader, wethAmount);
    }
```

Tool used

Manual Review

Recommendation

Consider using the `IWETH9` interface instead of `IERC20`.

Team

Implemented in https://github.com/opynfinance/squeeth-monorepo/pull/580.

Sherlock

Verified.

Issue I-5 Pre-increment in a loop is cheaper

Summary

Pre-increment in a loop is cheaper than post increment.

Severity

Informational

Vulnerability Detail

The loop `_orders` uses `i++` which costs more gas than `++i`.

Impact

Gas improvement.

Code Snippet

CrabStrategyV2.solL676

```
for (uint256 i = 0; i < _orders.length; i++)
```

Tool used

Manual Review

Recommendation

Use `++i` instead of `i++` to increment the value of a uint variable.

Team

Fixed with: https://github.com/opynfinance/squeeth-monorepo/pull/571.

Sherlock

Verified.

Issue I-6 No need to initialize variables with default values

Summary

Initialization to 0 or false is not necessary.

Severity

Informational

Vulnerability Detail

Initialization to 0 or false is not necessary, as these are the default values in Solidity.

Impact

This saves some gas.

Code Snippet

CrabStrategyV2.solL676

```
for (uint256 i = 0; i < _orders.length; i++)
```

Tool used

Manual Review

Recommendation

Remove the initialization values of 0 or false.

Team

Implemented in https://github.com/opynfinance/squeeth-monorepo/pull/575.

Sherlock

Verified.

Issue I-7 Use a constant for number values

Summary

In the several locations, `1e18` has been used instead of a pre-defined constant.

Severity

Informational

Vulnerability Detail

In the several locations, `1e18` has been used instead of a pre-defined constant. The private constant variable `ONE` can be used instead.

Impact

Improve readability.

Code Snippet

CrabStrategyV2.solL788, CrabStrategyV2.solL823, CrabStrategyV2.solL626

Tool used

Manual Review

Recommendation

Utilize constants for numbers used throughout the code.

Team

Implemented in https://github.com/opynfinance/squeeth-monorepo/pull/570.

Sherlock

Verified.

Issue I-8 Approve `EULER_MAINNET` in the constructor

Summary

`Weth` is approved to `EULER_MAINNET` on the onDeferredLiquidityCheck function. However, the approve can be called in the constructor.

Severity

Informational

Vulnerability Detail

`onDeferredLiquidityCheck` function approves the `EULER_MAINNET` address as spender for the WETH. When the `onDeferredLiquidityCheck` function is called it will always approve with `uint(max)`. Moving the approval into the constructor can save gas.

Impact

Gas improvement.

Code Snippet

CrabMigration.solL188

```
function onDeferredLiquidityCheck(bytes memory encodedData) external
↪   afterInitialized {
    require(msg.sender == EULER_MAINNET, "M4");

    FlashloanCallbackData memory data = abi.decode(encodedData,
↪   (FlashloanCallbackData));

    // 1. Borrow weth
    IDToken(dToken).borrow(0, data.amountToBorrow);
    weth.withdraw(data.amountToBorrow);

    // 2. Callback
    _flashCallback(data.caller, data.amountToBorrow, data.callSource,
↪   data.callData);

    // 4. Repay the weth:
    weth.deposit{value: data.amountToBorrow}();
    weth.approve(EULER_MAINNET, type(uint256).max);
    IDToken(dToken).repay(0, data.amountToBorrow);
}
```

Tool used

Manual Review

Recommendation

Move approval to constructor.

Team

Implemented in https://github.com/opynfinance/squeeth-monorepo/pull/569.

Sherlock

Verified.

Issue I-9 Unused state variable

Summary

During the code review, It has been noticed that `wPowerPerp` variable is set on the `CrabHelper` contract's constructor. However, the variable is not used in the contract.

Severity

Informational

Vulnerability Detail

During the code review, It has been noticed that `wPowerPerp` variable is set on the `CrabHelper` contract's constructor. However, the variable is not used in the contract.

Impact

Redundant variable

Code Snippet

CrabHelper.solL48

```
constructor(address _crab, address _swapRouter) StrategySwap(_swapRouter) {
    require(_crab != address(0), "Invalid crab address");

    crab = _crab;
    weth = ICrabStrategyV2(_crab).weth();
    wPowerPerp = ICrabStrategyV2(_crab).wPowerPerp();
}
```

Tool used

Manual Review

Recommendation

Delete unused state variable.

Team

Implemented in https://github.com/opynfinance/squeeth-monorepo/pull/568.

Sherlock

Verified.

Issue I-10 Prevent `transferFrom()`/`transfer()` with zero amount

Summary

Prevent `transferFrom()`/`transfer()` with zero amount.

Severity

Informational

Vulnerability Detail

The function `depositV1Shares()` does a `transferFrom()` transaction if `amount==0`, this would cost unnecessary gas.

Impact

Unnecessary gas.

Code Snippet

CrabMigration.solL141

```
function depositV1Shares(uint256 amount) external afterInitialized
↪   beforeMigration {
    sharesDeposited[msg.sender] += amount;
    totalCrabV1SharesMigrated += amount;
    crabV1.transferFrom(msg.sender, address(this), amount);
}
```

Tool used

Manual Review

Recommendation

Consider checking the `amount!=0`.

Team

We prefer to save small amounts of gas for most users who deposit with a non-zero amount vs saving gas for users who incorrectly deposit 0 shares.

Sherlock

Acknowledged.

Issue I-11 `sqrtPriceLimitX96` is ignored on the `_swapExactInputSingle` function

Summary

On the `_swapExactInputSingle`, `sqrtPriceLimitX96` parameter is defined as 0.

Severity

Informational

Vulnerability Detail

In the router, `sqrtPriceLimitX96` can be used to determine limits on the pool prices which cannot be exceeded by the swap. If it is set to zero, the parameter functionality is ignored.

Impact

According to Uniswap, in production this value can be used to set the limit for the price the swap will push the pool to, which can help protect against price impact or for setting up logic in a variety of price-relevant mechanisms.

Code Snippet

StrategySwap.solL28

```
function _swapExactInputSingle(
    address _tokenIn,
    address _tokenOut,
    address _from,
    address _to,
    uint256 _amountIn,
    uint256 _minAmountOut,
    uint24 _fee
) internal returns (uint256 amountOut) {
    // _from must approve this contract

    // Transfer the specified amount of tokenIn to this contract.
    IERC20(_tokenIn).transferFrom(_from, address(this), _amountIn);

    // Approve the router to spend tokenIn.
    IERC20(_tokenIn).approve(address(swapRouter), _amountIn);

    // We also set the sqrtPriceLimitx96 to be 0 to ensure we swap our exact
↪   input amount.
```

```
    ISwapRouter.ExactInputSingleParams memory params =
 ↪  ISwapRouter.ExactInputSingleParams({
        tokenIn: _tokenIn,
        tokenOut: _tokenOut,
        fee: _fee,
        recipient: _to,
        deadline: block.timestamp,
        amountIn: _amountIn,
        amountOutMinimum: _minAmountOut,
        sqrtPriceLimitX96: 0
    });

    // The call to `exactInputSingle` executes the swap.
    amountOut = swapRouter.exactInputSingle(params);
}
```

Tool used

Manual Review

Recommendation

Review the external interaction and utilize `sqrtPriceLimitX96` for the price impact.

Team

Opyn Team states that they use the maxAmountIn or minAmountOut to control slippage and don't want to or need to to control the after swap pool price.

Sherlock

Acknowledged.

Issue I-12 Missing overflow protection on the `depositV1Shares` function

Summary

Function `depositV1Shares` in `CrabMigration.sol` computes two additions without overflow protection.

Severity

Informational

Vulnerability Detail

Function `depositV1Shares` in `CrabMigration.sol` computes two additions without overflow protection. Even if user will not keep a uint(max) crabV1 token, the protection mechanism can be placed in the function.

Impact

Overflow risk

Code Snippet

CrabMigration.solL138

```
function depositV1Shares(uint256 amount) external afterInitialized
↪   beforeMigration {
    sharesDeposited[msg.sender] += amount;
    totalCrabV1SharesMigrated += amount;
    crabV1.transferFrom(msg.sender, address(this), amount);
}
```

Tool used

Manual Review

Recommendation

As `StrategyMath` library is already imported, use `add` function instead of the native + operator.

Team

We don’t see a way this could overflow given the only source of the amount to deposit is crab v1 tokens and this was an intentional choice based on that.

Sherlock

Agree on the comment, there is no way to exploit overflow. It marked as an acknowledged.

Issue I-13 Revert string size optimization

Summary

Shortening revert strings to fit in 32 bytes will decrease deploy time gas and will decrease runtime gas when the revert condition has been met.

Severity

Informational

Vulnerability Detail

Revert strings more than 32 bytes require at least one additional `mstore`, along with additional operations for computing memory offset.

Impact

Shortening revert strings to fit in 32 bytes will decrease deploy time gas and will decrease runtime gas when the revert condition has been met.

Code Snippet

CrabStrategyV2.solL598

```
...
        if (_order.isBuying) {
            require(_clearingPrice <= _order.price, "Clearing Price should be
↪   below bid price");
        } else {
            require(_clearingPrice >= _order.price, "Clearing Price should be
↪   above offer price");
        }
...
```

Tool used

Manual Review

Recommendation

Shorten the revert strings to fit in 32 bytes. Alternatively, the code could be modified to use custom errors, introduced in Solidity.

Team

Fixed in https://github.com/opynfinance/squeeth-monorepo/pull/563.

Sherlock

Verified.

Issue I-14 Optimize unsigned integer comparison

Summary

The check `!=0` costs less gas compared to `>0` for unsigned integers in require statements with the optimizer enabled.

Severity

Informational

Vulnerability Detail

The check `!=0` costs less gas compared to `>0` for unsigned integers in require statements with the optimizer enabled. While it may seem that `>0` is cheaper than `!=0`, this is only true without the optimizer enabled and outside a require statement. If the optimizer is enabled at `10k` and it is in a require statement, that would be more gas efficient.

Impact

`!=0` costs less gas compared to `>0` for unsigned integers.

Code Snippet

CrabStrategyV2.solL372, CrabStrategyV2.solL360, CrabStrategyV2.solL459, CrabStrategyV2.solL649, CrabStrategyV2.solL165, CrabStrategyV2.solL166

Tool used

Manual Review

Recommendation

Change `>0` comparison with `!=0`.

Team

We may consider this optimization in future code.

Sherlock

Acknowledged.

Issue I-15 `hedgePriceThreshold` does not have a sanity check

Summary

During the code review, it was observed that `hedgePriceThreshold` is scaled by `1e18`. However, there is no validation on the setter function.

Severity

Informational

Vulnerability Detail

On the `CrabStrategyV2` contract, the `hedgePriceThreshold` variable determines the deviation in wPowerPerp price that can trigger a rebalance. It is scaled by `1e18`, but there is no validation on the setter function.

Impact

That can lead to unexpected behavior on the hedging price calculation.

Code Snippet

CrabStrategyV2.solL371

```
function setHedgePriceThreshold(uint256 _hedgePriceThreshold) external onlyOwner
 ↪  {
    require(_hedgePriceThreshold > 0, "invalid hedge price threshold");

    hedgePriceThreshold = _hedgePriceThreshold;

    emit SetHedgePriceThreshold(_hedgePriceThreshold);
}
```

Tool used

Manual Review

Recommendation

Even if there is an owner check, consider adding the following check to ensure it's scaled by `1e18`.

```
require(_hedgePriceThreshold <= 1e18)
```

Team

Fixed here: https://github.com/opynfinance/squeeth-monorepo/pull/566.

Sherlock

Verified.

Issue I-16 State variable could be declared constant

Summary

State variables that never change can be declared constant. This can greatly reduce gas costs.

Severity

Informational

Vulnerability Detail

`maxOTCPriceTolerance` variable does not change on the contract. The variable can marked as a constant.

Impact

Reduce gas cost.

Code Snippet

CrabStrategyV2.solL55

```
uint256 public maxOTCPriceTolerance = 2e17; // 20%
```

Tool used

Manual Review

Recommendation

Add the constant keyword for state variables whose value never change.

Team

Implemented in https://github.com/opynfinance/squeeth-monorepo/pull/564.

Sherlock

Verified.

Issue I-17 Redundant Require Statement

Summary

`uint256` variables are initialized to a default value of zero per Solidity. However, in the `CrabStrategyV2` contract, variables are checked with the redundant require statement.

Severity

Informational

Vulnerability Detail

`uint256` variable are initialized to a default value of zero per Solidity. In the `initialize` function, the `require` statement is unnecessary because the values are already initialized with zero.

Impact

This saves some gas.

Code Snippet

CrabStrategyV2.solL213

```
...
        uint256 strategyDebt;
        uint256 strategyCollateral;

        _checkStrategyCap(amount, strategyCollateral);

        require((strategyDebt == 0 && strategyCollateral == 0), "C5");
...
```

Tool used

Manual Review

Recommendation

Consider removing the require statement.

Team

Fixed in https://github.com/opynfinance/squeeth-monorepo/pull/563.

Sherlock

Verified.