PlanetFinance

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **June 7th, 2021 - June 22nd, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	06/21/2021	Gabi Urrutia
0.1	Document Edits	06/22/2021	Souhail Mssassi
0.5	Document Edits	06/22/2021	Gabi Urrutia
1.0	Final Version	06/22/2021	Gabi Urrutia
1.1	Remediation Plan	06/25/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Souhail Mssassi	Halborn	Souhail.Mssassi@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

PlanetFinance engaged Halborn to conduct a security assessment on their Smart contract beginning on June 7th, 2021 and ending June 22nd, 2021. The security assessment was scoped to the smart contract repository. An audit of the security risk and implications regarding the changes introduced by the development team at PlanetFinance prior to its production release shortly following the assessments deadline.

The result of the audit is that some essential issues must be fixed.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned two full time security engineers to audit the security of the smart contract. The security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that smart contract functions are intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified several security risks, and recommends performing further testing to validate extended safety and correctness in context to the whole of contract. External threats, such as economic attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit.While manual testing is recommended to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose
- Smart Contract manual code read and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)

- Manual Assessment of use and safety for the critical solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Scanning of solidity files for vulnerabilities, security hotspots, or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (REMIX)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating

a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:
The security assessment was scoped to the smart contracts:

- AquaFarm.sol.
- AquaStrategy_4BELT.sol.
- AquaStrategy_AQUA.sol.
- AquaStrategy_PCS.sol.
- AquaToken.sol.
- PlanetFactory.sol.
- PlanetRouter.sol.
- TimelockController.sol.

Commit-ID: ac46bcadc5aacd291fbc8e8fe7b4bfe1766df6e8

OUT-OF-SCOPE:
Other smart contracts in the repository, external libraries and economics attacks.

However, if any economic issue is found, it will be marked as an INFORMATIONAL. This report identified several items that are economic in nature, (such as the way Liquidity can be accessed by owners) but may not be considered vulnerabilities in the context for this scope.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	10	6

LIKELIHOOD

IMPACT

	(HAL-01) (HAL-02)			
(HAL-06)				
(HAL-10)	(HAL-03) (HAL-05) (HAL-07)			
(HAL-13) (HAL-14) (HAL-15) (HAL-16) (HAL-17)	(HAL-08) (HAL-09) (HAL-11) (HAL-12)	(HAL-04)		
(HAL-18)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - TRANSFERRED AMOUNT VERIFICATION MISSING	Medium	MITIGATED (LOW) - 06/24/2021
HAL02 - OWNER CAN RENOUNCE OWNERSHIP	Medium	SOLVED - 06/24/2021
HAL03 - FLOATING PRAGMA	Low	SOLVED - 06/24/2021
HAL04 - LACK OF MINIMUM THRESHOLD	Low	ACKNOWLEDGED
HAL05 - FEE LIMIT DEFINITION MISSING	Low	ACKNOWLEDGED
HAL06 - USE OF BLOCK.TIMESTAMP	Low	RISK ACCEPTED
HAL07 - FOR LOOP OVER DYNAMIC ARRAY	Low	ACKNOWLEDGED
HAL08 - ADDRESS VALIDATION MISSING	Low	ACKNOWLEDGED
HAL09 - MISSING EVENT HANDLER	Low	RISK ACCEPTED
HAL10 - IGNORED RETURN VALUES	Low	RISK ACCEPTED
HAL11 - MULTIPLE PRAGMA DEFINITION	Low	FUTURE RELEASE UPDATE
HAL12 - EXPERIMENTAL FEATURES ENABLED	Low	RISK ACCEPTED
HAL13 - LACK OF LIQUIDITY LOSS PROTECTION	Informational	ACKNOWLEDGED
HAL14 - USE OF INLINE ASSEMBLY	Informational	RISK ACCEPTED
HAL15 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	RISK ACCEPTED
HAL16 - USE OF LOW-LEVEL CALLS	Informational	RISK ACCEPTED
HAL17 - NO TEST COVERAGE	Informational	FUTURE RELEASE UPDATE
HAL18 - DOCUMENTATION	Informational	FUTURE RELEASE UPDATE

FINDINGS & TECH DETAILS

3.1 (HAL-01) TRANSFERRED AMOUNT VERIFICATION MISSING - MEDIUM

Description:

In order to keep track of users' shares in pools, a corresponding amount of liquidity pool tokens is minted to liquidity providers. The exact amount to be minted is calculated based on the declared amount of ERC20 tokens added to the pool.

In the addLiquidityPair function PlanetFinance use safeTransferFrom from the TransferHelper library to handle the token transfer. This function calls transferFrom in the token contract to actually execute the transfer. However, since the actual amount transferred ie. the delta of previous (before transfer) and current (after transfer) balance is not verified, a malicious user may list a custom ERC20 token with the transferFrom function modified in such a way that it does not transfer any tokens at all and the attacker is still going to have their liquidity pool tokens minted anyway.

Code Location:

Attacker-controlled example ERC20 token contract

Listing 1: EvilERC20.sol (Lines 10)

```
1  function transferFrom(
2      address from,
3      address to,
4      uint256 value
5    )
6      public
7      override
8      returns (bool)
9    {
10     value = 1;
11     require(value <= _balances[from]);
12     require(value <= _allowed[from][msg.sender]);
```

```
13      require(to != address(0));
14
15      _balances[from] = _balances[from].sub(value);
16      _balances[to] = _balances[to].add(value);
17      _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(
            value);
18      emit Transfer(from, to, value);
19      return true;
20    }
```

PlanetRouter.sol Line #23

Listing 2: PlanetRouter.sol (Lines 25)

```
23 function safeTransferFrom(address token, address from, address to,
       uint value) internal {
24      // bytes4(keccak256(bytes('transferFrom(address,address,
            uint256)')));
25      (bool success, bytes memory data) = token.call(abi.
            encodeWithSelector(0x23b872dd, from, to, value));
26      require(success && (data.length == 0 || abi.decode(data, (bool
            ))), 'TransferHelper: TRANSFER_FROM_FAILED');
27  }
```

PlanetRouter.sol Line #442

Listing 3: PlanetRouter.sol (Lines 456)

```
442 function addLiquidity(
443     address tokenA,
444     address tokenB,
445     uint amountADesired,
446     uint amountBDesired,
447     uint amountAMin,
448     uint amountBMin,
449     address to,
450     uint deadline
451 ) external virtual override ensure(deadline) returns (uint amountA
       , uint amountB, uint liquidity) {
452     (amountA, amountB) = _addLiquidity(tokenA, tokenB,
            amountADesired, amountBDesired, amountAMin, amountBMin);
453     address pair = PlanetLibrary.pairFor(factory, tokenA, tokenB);
```

```
454     TransferHelper.safeTransferFrom(tokenA, msg.sender, pair,
            amountA);
455     TransferHelper.safeTransferFrom(tokenB, msg.sender, pair,
            amountB);
456     liquidity = IPlanetPair(pair).mint(to);
457 }
```

Risk Level:

Likelihood - 2
Impact - 5

Recommendation:

Whenever tokens are transferred, the delta of the previous (before transfer) and current (after transfer) token balance should be verified to match the user-declared token amount.

Remediation Plan:

MITIGATED: Planet.Finance team proposes remediate this by listing only those pairs on their website which adhere to ERC20 standard and are verified by them. Thus this is not an issue within the contract and also not a threat to the users investment unless they do trade off the website. The issue is reclassified as LOW.

3.2 (HAL-02) OWNER CAN RENOUNCE OWNERSHIP - MEDIUM

Description:

The Owner of the contract is usually the account which deploys the contract. As a result, the Owner is able to perform some privileged actions. In the AquaStrategy-4BELT.sol smart contracts, the renounceOwnership function is used to renounce being Owner. Otherwise, if the ownership was not transferred before, the contract will never have an Owner, which is dangerous.

Code Location:

AquaStrategy-PCS.sol Line #1

Listing 4: AquaStrategy-PCS.sol (Lines 1335)

```
1335 function renounceOwnership() public virtual onlyOwner {
1336         emit OwnershipTransferred(_owner, address(0));
1337         _owner = address(0);
1338     }
```

AquaStrategy-4BELT.sol Line #1

Listing 5: AquaStrategy-4BELT.sol (Lines 1326)

```
1326 function renounceOwnership() public virtual onlyOwner {
1327         emit OwnershipTransferred(_owner, address(0));
1328         _owner = address(0);
1329     }
```

AquaToken.sol Line #1

Listing 6: AquaToken.sol (Lines 636)

```
636 function renounceOwnership() public virtual onlyOwner {
637         emit OwnershipTransferred(_owner, address(0));
638         _owner = address(0);
639     }
```

Risk Level:

Likelihood - 2
Impact - 5

Recommendation:

It's recommended that the Owner is not able to call renounceOwnership without transferring the Ownership to other address before. In addition, if a multi-signature wallet is used, calling renounceOwnership function should be confirmed for two or more users. As an other solution, Renounce Ownership functionality can be disabled with the following line.
AquaStrategy-4BELT.sol Line #1

Listing 7: AquaStrategy-4BELT.sol (Lines 3)

```
2 function renounceOwnership () public override onlyOwner {
3     revert ("can 't renounceOwnership here "); // not possible
          with this smart contract
4 }
```

Remediation Plan:

SOLVED: In the constructor parameter, the ownership is transferred to the farm contract at the time of deployment and the farm contract cannot call this function. In addition, Planet.Finance team implements multi-signature wallets.

3.3 (HAL-03) FLOATING PRAGMA - LOW

Description:

PlanetRouter.sol contract use the floating pragma >=0.5.0. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the **pragma** helps to ensure that contracts do not accidentally get deployed using another pragma, for example, either an outdated pragma version that might introduce bugs that affect the contract system negatively or a recently released pragma version which has not been extensively tested.

Reference: ConsenSys Diligence - Lock pragmas

Code Location:

PlanetRouter.sol Line #1

Listing 8: PlanetLibrary.sol (Lines 277)

```
277 pragma solidity >=0.5.0;
```

- This is an example where the floating pragma is used. ^0.5.0.

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Consider lock the pragma version known bugs for the compiler version. Therefore, it is recommended not to use floating pragma in the production. Apart from just locking the pragma version in the code, the sign (>=) need to be removed. it is possible locked the pragma fixing the version both in truffle-config.js if you use the Truffle framework and

in hardhat.config.js if you use HardHat framework for the deployment.

Remediation Plan:

SOLVED: Pragma will be locked before deploying the contract.

3.4 (HAL-04) LACK OF MINIMUM THRESHOLD - LOW

Description:

When modifying the MinTimeToWithdraw variable, a check is made in the code which is that the new value must always be smaller than the old value, the problem here is that this variable will always be decreasing and if by mistake the value 0 is injected we can't modify the value of newMinTimeToWithdraw by a value greater than 0.

Code Location:

AquaStrategy-AQUA.sol Line #1

Listing 9: AquaStrategy-AQUA.sol (Lines 2419)

```
2419            require(newMinTimeToWithdraw <= minTimeToWithdrawUL, "too
                    high");
2420            emit minTimeToWithdrawChanged(minTimeToWithdraw,
                    newMinTimeToWithdraw);
2421            minTimeToWithdraw = newMinTimeToWithdraw
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

It is recommended performing a limit after the verification is done to check the new value and compare it with 900 seconds which is the value where the miner can manipulate the block.timestamp

Remediation Plan:

ACKNOWLEDGED: Planet.Finance team considers that this is the expected behaviour, the lower the threshold the better for investors.

3.5 (HAL-05) FEE LIMIT DEFINITION MISSING - LOW

Description:

During the tests, Halborn Team noticed that on the _mintFee function, limits are not defined.

Code Location:

PlanetFactory.sol Line #338

Listing 10: PlanetFactory.sol (Lines 338)

```
338     function _mintFee(uint112 _reserve0, uint112 _reserve1)
            private returns (bool feeOn) {
339         address feeTo = IPlanetFactory(factory).feeTo();
340         feeOn = feeTo != address(0);
341         uint _kLast = kLast; // gas savings
342         if (feeOn) {
343             if (_kLast != 0) {
344                 uint rootK = Math.sqrt(uint(_reserve0).mul(
                        _reserve1));
345                 uint rootKLast = Math.sqrt(_kLast);
346                 if (rootK > rootKLast) {
347                     uint numerator = totalSupply.mul(rootK.sub(
                            rootKLast)).mul(12);
348                     uint denominator = rootK.mul(13).add(rootKLast
                            .mul(12));
349                     uint liquidity = numerator / denominator;
350                     if (liquidity > 0) _mint(feeTo, liquidity);
351                 }
352             }
353         } else if (_kLast != 0) {
354             kLast = 0;
355         }
356     }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It is recommended define maximum and minimum fee range on the related function.

Remediation Plan:

ACKNOWLEDGED: Planet.Finance team defines a fixed percentage of fees which goes to the developers and that depends on the transaction amount. They consider that a fee limit is not necessary.

3.6 (HAL-06) USE OF BLOCK.TIMESTAMP - LOW

Description:

The contracts PlanetFactory.sol, use block.timestamp. The global variable block.timestamp does not necessarily hold the current time, and may not be accurate. Miners can influence the value of block.timestamp to perform Maximal Extractable Value (MEV) attacks. There is no guarantee that the value is correct, only that it is higher than the previous block's timestamp.

Code Location:

PlanetFactory.sol Line #1

Listing 11: PlanetFactory.sol (Lines 188)

```
187 function permit(address owner, address spender, uint value, uint
      deadline, uint8 v, bytes32 r, bytes32 s) external {
188         require(deadline >= block.timestamp, 'Planet: EXPIRED');
189         bytes32 digest = keccak256(
```

PlanetRouter.sol Line #1

Listing 12: PlanetRouter.sol (Lines 400)

```
399   modifier ensure(uint deadline) {
400         require(deadline >= block.timestamp, 'PlanetRouter:
              EXPIRED');
401         _;
402     }
```

Recommendation:

Use block.number instead of block.timestamp or now to reduce the risk of MEV attacks. Check if the timescale of the project occurs across years,

days and months rather than seconds. If possible, it is recommended to use Oracles.

Remediation Plan:

RISK ACCEPTED: Planet.Finance team considers acceptable the use of `block.timestamp`.

3.7 (HAL-07) FOR LOOP OVER DYNAMIC ARRAY - LOW

Description:

Calls inside a loop might lead to a denial-of-service attack. The function discovered is a for loop on variable `pid` that iterates up to the poolInfo length . If this integer is evaluated at extremely large numbers,this can cause a DoS.

Code Location:

AquaFarm.sol Line #1

Listing 13: AquaFarm.sol (Lines 1568,1569)

```
1566 function massUpdatePools() public {
1567         uint256 length = poolInfo.length;
1568         for (uint256 pid = 0; pid < length; ++pid) {
1569             updatePool(pid);
1570         }
1571     }
```

PlanetRouter.sol Line #1

Listing 14: PlanetRouter.sol (Lines 341)

```
341 for (uint i; i < path.length - 1; i++) {
342             (uint reserveIn, uint reserveOut) = getReserves(
                    factory, path[i], path[i + 1]);
343             amounts[i + 1] = getAmountOut(amounts[i], reserveIn,
                    reserveOut);
344         } }
345     }
```

Listing 15: PlanetRouter.sol (Lines 352)

```
352  for (uint i = path.length - 1; i > 0; i--) {
353             (uint reserveIn, uint reserveOut) = getReserves(
                    factory, path[i - 1], path[i]);
354             amounts[i - 1] = getAmountIn(amounts[i], reserveIn,
                    reserveOut);
355         }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

If possible, use pull over push strategy for external calls.

Remediation Plan:

ACKNOWLEDGED: Planet.Finance team considers that pool length will never have a high enough.

3.8 (HAL-08) ADDRESS VALIDATION MISSING - LOW

Description:

Address validation is missing in many functions in which user supplied input is assigned to state variables directly. This could lead to irrecoverable loss of tokens or sensitive contract features.

Code Location:

AquaStrategy_4BELT.sol Line #2340

Listing 16: AquaStrategy_4BELT.sol (Lines 2342,2347,2348)

```
2340 function changeFeeAddressSetter(address payable
       _newFeeAddressSetter) public {
2341     require(_msgSender() == feeAddressesSetter,"Access Denied");
2342     feeAddressesSetter = _newFeeAddressSetter;
2343 }
2344
2345 function changeFeeAddress(address _depositFeeAddress,address
       _withdrawFeeAddress) public {
2346     require(_msgSender() == feeAddressesSetter,"Access Denied");
2347     depositFeeAddress = _depositFeeAddress;
2348     withdrawFeeAddress = _withdrawFeeAddress;
2349 }
```

Also functions changeFeeAddressSetter and ChangeFeeAddress in AquaStrategy_AQUA.sol and AquaStrategy_PCS.sol.

AquaStrategy_4BELT.sol Line #2247

Listing 17: AquaStrategy_4BELT.sol

```
2247 wbnbAddress = _addresses[0];
2248 govAddress = _addresses[1];
2249 aquaFarmAddress = _addresses[2];
2250 AQUAAddress = _addresses[3];
```

```
2251
2252 wantAddress = _addresses[4];
2253 token0Address = _addresses[5];
2254 token1Address = _addresses[6];
2255 earnedAddress = _addresses[7];
2256
2257 farmContractAddress = _addresses[8];
2258 pid = _pid;
2259 isCAKEStaking = _isCAKEStaking;
2260 isSameAssetDeposit = _isSameAssetDeposit;
2261 isAquaComp = _isAquaComp;
2262
2263 uniRouterAddress = _addresses[9];
2264 earnedToAQUAPath = _earnedToAQUAPath;
2265 earnedToToken0Path = _earnedToToken0Path;
2266 earnedToToken1Path = _earnedToToken1Path;
2267 token0ToEarnedPath = _token0ToEarnedPath;
2268 token1ToEarnedPath = _token1ToEarnedPath;
2269
2270 controllerFee = _controllerFee;
2271 rewardsAddress = _addresses[10];
2272 buyBackRate = _buyBackRate;
2273 buyBackAddress = _addresses[11];
2274 entranceFeeFactor = _entranceFeeFactor;
2275 withdrawFeeFactor = _withdrawFeeFactor;
2276
2277 belt4PoolAddress = _belt4PoolAddress;
2278 feeAddressesSetter = _msgSender();
2279 transferOwnership(aquaFarmAddress);
```

Also constructor functions in changeFeeAddressSetter and ChangeFeeAddress in AquaStrategy _AQUA.sol and AquaStrategy_PCS.sol.

AquaFarm.sol Line #1720

Listing 18: AquaFarm.sol (Lines 1721)

```
1720 function changeAQUAaddress(address _newAddress) public onlyOwner {
1721     AQUA = _newAddress;
1722 }
```

PlanetRouter.sol Line #404

Listing 19: PlanetRouter.sol (Lines 405,406)

```
404 constructor(address _factory, address _WETH) public {
405     factory = _factory;
406     WETH = _WETH;
407 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Add proper address validation whenever user-supplied input is assigned to state variables. Ideally, all input should be validated against whitelists. Also, consider implementing relevant setter functions for particularly sensitive variables.

Remediation Plan:

ACKNOWLEDGED: Planet.Finance team considers that if this happens, they still can change the fees as they can make low level calls through the timelock.

3.9 (HAL-09) MISSING EVENT HANDLER - LOW

Description:

In the Planet.Finance contract the some of functions do not emit event after the progress. Events are a method of informing the transaction initiator about the actions taken by the called function. It logs its emitted parameters in a specific log history, which can be accessed outside of the contract using some filter parameters.

PlanetFactory.sol Line #~488

Listing 20: PlanetFactory.sol (Lines 488,493)

```
488     function setFeeTo(address _feeTo) external {
489         require(msg.sender == feeToSetter, 'Planet: FORBIDDEN');
490         feeTo = _feeTo;
491     }
492
493     function setFeeToSetter(address _feeToSetter) external {
494         require(msg.sender == feeToSetter, 'Planet: FORBIDDEN');
495         feeToSetter = _feeToSetter;
496     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider as much as possible declaring events at the end of function. Events can be used to detect the end of the operation.

Remediation Plan:

RISK ACCEPTED: Planet.Finance team accepts the risk.

3.10 (HAL-10) IGNORED RETURN VALUES - LOW

Description:

The return value of an external call is not stored in a local or state variable. In the Planet.Finance contract, there are a few instances where the multiple methods are called and the return value (bool) is ignored.

PlanetFactory.sol Line #~414,484
AquaFarm.sol Line #~1685,1710
AquaStrategy_AQUA.sol Line #~1821,1879,1973-1982,2317,2352
AquaStrategy_4BELT.sol Line #~1821,1879,1973-1982,2317,2352
AquaStrategy_PCS.sol Line #~1821,1879,1973-1982,2317,2352

Listing 21: PlanetFactory.sol (Lines 414)

```
414     function _addLiquidity(
415         address tokenA,
416         address tokenB,
417         uint amountADesired,
418         uint amountBDesired,
419         uint amountAMin,
420         uint amountBMin
421     ) internal virtual returns (uint amountA, uint amountB) {
422         // create the pair if it doesn't exist yet
423         if (IPlanetFactory(factory).getPair(tokenA, tokenB) ==
                address(0)) {
424             IPlanetFactory(factory).createPair(tokenA, tokenB);
425         }
426         (uint reserveA, uint reserveB) = PlanetLibrary.getReserves
                (factory, tokenA, tokenB);
427         if (reserveA == 0 && reserveB == 0) {
428             (amountA, amountB) = (amountADesired, amountBDesired);
429         } else {
430             uint amountBOptimal = PlanetLibrary.quote(
                    amountADesired, reserveA, reserveB);
431             if (amountBOptimal <= amountBDesired) {
432                 require(amountBOptimal >= amountBMin, '
                        PlanetRouter: INSUFFICIENT_B_AMOUNT');
433                 (amountA, amountB) = (amountADesired,
```

```
                    amountBOptimal);
434             } else {
435                 uint amountAOptimal = PlanetLibrary.quote(
                        amountBDesired, reserveB, reserveA);
436                 assert(amountAOptimal <= amountADesired);
437                 require(amountAOptimal >= amountAMin, '
                        PlanetRouter: INSUFFICIENT_A_AMOUNT');
438                 (amountA, amountB) = (amountAOptimal,
                        amountBDesired);
439             }
440         }
441     }
```

Listing 22: PlanetFactory.sol (Lines 484)

```
484     function removeLiquidity(
485         address tokenA,
486         address tokenB,
487         uint liquidity,
488         uint amountAMin,
489         uint amountBMin,
490         address to,
491         uint deadline
492     ) public virtual override ensure(deadline) returns (uint
            amountA, uint amountB) {
493         address pair = PlanetLibrary.pairFor(factory, tokenA,
                tokenB);
494         IPlanetPair(pair).transferFrom(msg.sender, pair, liquidity
                );
495         .....
496         require(amountA >= amountAMin, 'PlanetRouter:
                INSUFFICIENT_A_AMOUNT');
497         require(amountB >= amountBMin, 'PlanetRouter:
                INSUFFICIENT_B_AMOUNT');
498     }
```

Listing 23: PlanetFactory.sol (Lines 1694,1706,1708)

```
1685     function emergencyWithdraw(uint256 _pid) public nonReentrant {
1686         PoolInfo storage pool = poolInfo[_pid];
1687         UserInfo storage user = userInfo[_pid][msg.sender];
1688 
1689         uint256 wantLockedTotal =
```

```
690             IStrategy(poolInfo[_pid].strat).wantLockedTotal();
691         uint256 sharesTotal = IStrategy(poolInfo[_pid].strat).
                sharesTotal();
692         uint256 amount = user.shares.mul(wantLockedTotal).div(
                sharesTotal);
693
694         IStrategy(poolInfo[_pid].strat).withdraw(msg.sender,
                amount);
695
696         pool.want.safeTransfer(address(msg.sender), amount);
697         emit EmergencyWithdraw(msg.sender, _pid, amount);
698         user.shares = 0;
699         user.rewardDebt = 0;
700     }
701
702     // Safe AQUA transfer function, just in case if rounding error
            causes pool to not have enough
703     function safeAQUATransfer(address _to, uint256 _AQUAAmt)
            internal {
704         uint256 AQUABal = IERC20(AQUA).balanceOf(address(this));
705         if (_AQUAAmt > AQUABal) {
706             IERC20(AQUA).transfer(_to, AQUABal);
707         } else {
708             IERC20(AQUA).transfer(_to, _AQUAAmt);
709         }
710     }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Add a return value check to avoid an unexpected crash of the contract. Return value checks provide better exception handling.

Remediation Plan:

RISK ACCEPTED: Planet.Finance team accepts the risk.

3.11 (HAL-11) MULTIPLE PRAGMA DEFINITION - LOW

Description:

In the Planet.Finance contracts, Pragma version is defined multiple times. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the **pragma** helps to ensure that contracts do not accidentally get deployed using another pragma, for example, either an outdated pragma version that might introduce bugs that affect the contract system negatively or a recently released pragma version which has not been extensively tested.

TimelockController.sol Line #~7,703,955,979

Listing 24: TimelockController.sol (Lines)

```
 7 pragma solidity 0.6.12;
 8 pragma solidity >=0.6.2 <0.8.0;
 9 pragma solidity >=0.6.0 <0.8.0;
10 pragma solidity >=0.6.0 <0.8.0;
```

PlanetRouter.sol Line #~ 7,37,135,180,202,222,277,361,381,391

Listing 25: PlanetRouter.sol (Lines)

```
 7 pragma solidity >=0.6.0;
 8 pragma solidity >=0.6.2;
 9 pragma solidity >=0.6.2;
10 pragma solidity >=0.5.0;
11 pragma solidity >=0.5.0;
12 pragma solidity >=0.5.0;
13 pragma solidity >=0.5.0;
14 pragma solidity >=0.5.0;
15 pragma solidity >=0.5.0;
16 pragma solidity =0.6.6;
17
```

PlanetRouter.sol Line #~ 7,37,135,180,202,222,277,361,381,391

Listing 26: PlanetRouter.sol (Lines)

```
 7 pragma solidity >=0.6.0;
 8 pragma solidity >=0.6.2;
 9 pragma solidity >=0.6.2;
10 pragma solidity >=0.5.0;
11 pragma solidity >=0.5.0;
12 pragma solidity >=0.5.0;
13 pragma solidity >=0.5.0;
14 pragma solidity >=0.5.0;
15 pragma solidity >=0.5.0;
16 pragma solidity =0.6.6;
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider lock and use single pragma version known bugs for the compiler version.

Remediation Plan:

PENDING: Planet.Finance team will fix it in a future release.

3.12 (HAL-12) EXPERIMENTAL FEATURES ENABLED - LOW

Description:

ABIEncoderV2 is enabled to be able to pass struct type into a function both web3 and another contract. The use of experimental features could be dangerous on live deployments. The experimental ABI encoder does not handle non-integer values shorter than 32 bytes properly. This applies to bytesNN types, bool, enum and other types when they are part of an array or a struct and encoded directly from storage. This means these storage references have to be used directly inside abi.encode(. . .) as arguments in external function calls or in event data without prior assignment to a local variable. Using return does not trigger the bug. The types bytesNN and bool will result in corrupted data while enum might lead to an invalid revert.

Furthermore, arrays with elements shorter than 32 bytes may not be handled correctly even if the base type is an integer type. Encoding such arrays in the way described above can lead to other data in the encoding being overwritten if the number of elements encoded is not a multiple of the number of elements that fit a single slot. If nothing follows the array in the encoding (note that dynamically-sized arrays are always encoded after statically-sized arrays with statically-sized content), or if only a single array is encoded, no other data is overwritten. There are known bugs that are publicly released while using this feature. However, the bug only manifests itself when all the following conditions are met:

- Storage data involving arrays or structs is sent directly to an external function call, to abi.encode or to event data without prior assignment to a local (memory) variable.
- There is an array that contains elements with size less than 32 bytes or a struct that has elements that share a storage slot or members of type bytesNN shorter than 32 bytes. In addition to that, in the following situations, your code is NOT affected:
- All the structs or arrays only use uint256 or int256 types. If you

only use integer types (that may be shorter) and only encode at most one array at a time. If you only return such data and do not use it in abi.encode, external calls or event data.

Reference: https://blog.ethereum.org/2019/03/26/solidity-optimizer-and-abiencoderv2

ABIEncoderV2 is enabled to be able to pass struct type into a function both web3 and another contract. Naturally, any bug can have wildly varying consequences depending on the program control flow, but we expect that this is more likely to lead to malfunction than exploitability. The bug, when triggered, will under certain circumstances send corrupt parameters on method invocations to other contracts.

TimelockController.sol Line #~8

Listing 27: TimelockController.sol (Lines 8)

```
7 pragma solidity 0.6.12;
8 pragma experimental ABIEncoderV2;
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

When possible, do not use experimental features in the final live deployment. Validate and check that all the conditions above are true for integers and arrays (i.e. all using uint256).

Remediation Plan:

RISK ACCEPTED: Planet.Finance team accepts the risk.

3.13 (HAL-13) LACK OF LIQUIDITY LOSS PROTECTION - INFORMATIONAL

Description:

In the TimelockController.sol, Halborn team noticed that withdraw and changeAQUAaddress progress do not have any timelock protection mechanisms. Additionaly, the inCaseTokensGetStuck function in AquaFarm.sol,TimelockController.sol, AquaStrategy_4BELT.sol, AquaStrategy_AQUA.sol and AquaStrategy_PCS.sol allows the contract owner to transfer the deposited tokens to their account. These situations are often enabled because a single executor role, or a liquidity address has access to remove all the TVL (Total Value Locked) through a withdraw or transfer function. While sometimes, the developer or owner does not intend to do this malicious act, the risk still exists if the private key is stolen since there is nothing preventing the key-holder from calling the withdraw.

Code Location:

TimelockController.sol Line #1783

Listing 28: TimelockController.sol (Lines 1783,1788)

```
1783     function withdrawBNB() public payable {
1784         require(msg.sender == devWalletAddress, "!devWalletAddress
                ");
1785         devWalletAddress.transfer(address(this).balance);
1786     }
1787 
1788     function withdrawBEP20(address _tokenAddress) public payable {
1789         require(msg.sender == devWalletAddress, "!devWalletAddress
                ");
1790         uint256 tokenBal = IERC20(_tokenAddress).balanceOf(address
                (this));
1791         IERC20(_tokenAddress).safeIncreaseAllowance(
                devWalletAddress, tokenBal);
1792         IERC20(_tokenAddress).transfer(devWalletAddress, tokenBal)
                ;
```

```
1793     }
```

AquaFarm.sol Line #1717

Listing 29: AquaFarm.sol (Lines 1717)

```
1712 function inCaseTokensGetStuck(address _token, uint256 _amount)
1713     public
1714     onlyOwner
1715 {
1716     require(_token != AQUA, "!safe");
1717     IERC20(_token).safeTransfer(msg.sender, _amount);
1718 }
```

TimelockController.sol Line #1872

Listing 30: TimelockController.sol (Lines 1878)

```
1872     function inCaseTokensGetStuck(
1873         address _stratAddress,
1874         address _token,
1875         uint256 _amount,
1876         address _to
1877     ) public virtual onlyRole(EXECUTOR_ROLE) {
1878         IStrategy(_stratAddress).inCaseTokensGetStuck(_token,
                _amount,_to);
1879     }
```

AquaStrategy_4BELT.sol Line #2174
AquaStrategy_AQUA.sol Line #2174
AquaStrategy_PCS.sol Line #2174

Listing 31: (Lines 2174)

```
2174 function inCaseTokensGetStuck(address _token, uint256 _amount)
2175     public
2176     onlyOwner
2177 {
2178     require(_token != AQUA, "!safe");
2179     IERC20(_token).safeTransfer(msg.sender, _amount);
```

```
2180 }
```

AquaFarm.sol Line #1720

Listing 32: (Lines 1720)

```
1720 function changeAQUAaddress(address _newAddress) public onlyOwner {
1721         AQUA = _newAddress;
1722     }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Those functions allows the executors or owners of the system to perform withdraw all amounts from token addresses. The owner should be limited to the minimum operations possible that allows pool management. PlanetFinance does only use the onlyOwner modifier , walletAddress and executor role check to perform critical actions such as enabling transfers on the tokens. However, these functionalities should be split between multiple role based users with multi-signature wallets for each one. Also, It is recommended that add timelock or pause/unpause functionality instead of transfer tokens. If it is not intended behaviour of the contracts, the codes should be deleted from the repository. As an another solution, the governance mechanism should be implemented on the critical changes.

Remediation Plan:

ACKNOWLEDGED: There is a condition which checks that no one can remove the pool tokens. It is only for tokens which are stuck accidentally. Thus the TVL is safe:

Listing 33

```
        require(_token != earnedAddress, "!safe");
        require(_token != wantAddress, "!safe");
```

3.14 (HAL-14) USE OF INLINE ASSEMBLY - INFORMATIONAL

Description:

Inline assembly is a way to access the Virtual Machine at a low level. This discards several important safety features in Solidity.

Code Location:

PlanetFactory.sol Line #1

Listing 34: PlanetFactory.sol (Lines 132,133)

```
130     constructor() public {
131         uint chainId;
132         assembly {
133             chainId := chainid
134         }
```

Listing 35: PlanetFactory.sol (Lines 478)

```
478 assembly {
479             pair := create2(0, add(bytecode, 32), mload(bytecode),
                    salt)
480         }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

The contracts should avoid using inline assembly because it interacts with the EVM (Ethereum Virtual Machine) at a low level. An attacker

could bypass many essential safety features of Solidity.

Remediation Plan:

RISK ACCEPTED: Planet.Finance team accepts the risk.

3.15 (HAL-15) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In the public functions, array arguments are immediately copied to memory, while external functions can read directly from calldata. Reading calldata is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments being located in memory when the compiler generates the code for an internal function.

Code Location:

We noticed the use of public functions in the following contract:

- AquaFarm.sol

Listing 36: AquaFarm.sol (Lines 298)

```
298         function symbol() public view returns (string memory) {
```

Listing 37: AquaFarm.sol (Lines 315)

```
315     function decimals() public view returns (uint8) {
```

Listing 38: AquaFarm.sol (Lines 298)

```
298         function symbol() public view returns (string memory) {
```

Listing 39: AquaFarm.sol (Lines 315)

```
315     function decimals() public view returns (uint8) {
```

Listing 40: AquaFarm.sol (Lines 322)

```
322     function totalSupply() public view override returns (uint256)
            {
```

Listing 41: AquaFarm.sol (Lines 329)

```
329     function balanceOf(address account) public view override
            returns (uint256) {
```

Listing 42: AquaFarm.sol (Lines 341)

```
341     function transfer(address recipient, uint256 amount)
```

Listing 43: AquaFarm.sol (Lines 354)

```
354     function allowance(address owner, address spender)
```

Listing 44: AquaFarm.sol (Lines 371)

```
371     function approve(address spender, uint256 amount)
```

Listing 45: AquaFarm.sol (Lines 394)

```
394     function transferFrom(
```

Listing 46: AquaFarm.sol (Lines 423)

```
423     function increaseAllowance(address spender, uint256 addedValue
            )
```

Listing 47: AquaFarm.sol (Lines 450)

```
450     function decreaseAllowance(address spender, uint256
            subtractedValue)
```

Listing 48: AquaFarm.sol (Lines 423)

```
423     function increaseAllowance(address spender, uint256 addedValue
            )
```

Listing 49: AquaStrategy-4BELT.sol (Lines 294)

```
294 function name() public view returns (string memory) {
```

Listing 50: AquaStrategy-4BELT.sol (Lines 302)

```
302 function symbol() public view returns (string memory) {
```

Listing 51: AquaStrategy-4BELT.sol (Lines 319)

```
319 function decimals() public view returns (uint8) {
```

Listing 52: AquaStrategy-4BELT.sol (Lines 326)

```
326 function totalSupply() public view override returns (uint256) {
```

Listing 53: AquaStrategy-AQUA.sol (Lines 294)

```
294 function name() public view returns (string memory) {
```

Listing 54: AquaStrategy-AQUA.sol (Lines 1326)

```
1326 function renounceOwnership() public virtual onlyOwner {
```

Listing 55: AquaStrategy-AQUA.sol (Lines 1326)

```
1326 function changeFeeAddressSetter(address payable
        _newFeeAddressSetter) public {
```

Risk Level:

Likelihood - 1

Impact - 2

Recommendation:

Consider declaring external variables instead of public variables. A best practice is to use external if expecting a function to only be called externally and public if called internally. Public functions are always accessible, but external functions are only available to external callers.

Remediation Plan:

RISK ACCEPTED: Planet.Finance team accepts the risk.

3.16 (HAL-16) USE OF LOW-LEVEL CALLS - INFORMATIONAL

Description:

Without checking the return value of a low-level message call, execution will continue even if the called contract throws an exception. If the call fails incidentally or an attacker induces the call to fail, the following software logic may result in unexpected consequences.

Code Location:

TimelockController.sol.sol Line #1

Listing 56: TimelockController.sol.sol (Lines 1677,1678)

```
1677         (bool success, ) = target.call{value: value}(data);
1678         require(success, "TimelockController: underlying
                transaction reverted");
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

If possible, it is recommended to avoid the use of low level calls.

Remediation Plan:

RISK ACCEPTED: Planet.Finance team accepts the risk.

3.17 (HAL-17) NO TEST COVERAGE - INFORMATIONAL

Description:

Unlike traditional software, smart contracts can not be modified unless deployed using a proxy contract. Because of the permanence, unit tests and functional testing are recommended to ensure the code works correctly before deployment. Mocha and Chai are valuable tools to perform unit tests in smart contracts. Mocha is a Javascript testing framework for creating synchronous and asynchronous unit tests, and Chai is a library with assertion functionality such as assert or expect and should be used to develop custom unit tests.

References:
https://github.com/mochajs/mocha
https://github.com/chaijs/chai
https://docs.openzeppelin.com/learn/writing-automated-tests

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

We recommend performing as many test cases as possible to cover all conceivable scenarios in the smart contract.

Remediation Plan:

PENDING: Planet.Finance team will fix it in a future release.

3.18 (HAL-18) DOCUMENTATION - INFORMATIONAL

Description:

The documentation provided by the PlanetFinance team is not complete. For instance, the documentation included in the GitHub repository should include a walkthrough to deploy and test the smart contracts.

Recommendation:

Consider updating the documentation in Github for greater ease when contracts are deployed and tested. Have a Non-Developer or QA resource work through the process to make sure it addresses any gaps in the set-up steps due to technical assumptions.

Remediation Plan:

PENDING: Planet.Finance team will fix it in a future release.

3.19 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

AquaFarm.sol

```
INFO:Detectors:
PlanetFinance.pendingAQUA(uint256,address) (AquaFarm.sol#1524-1545) performs a multiplication on the result of a division:
        -AQUAReward = multiplier.mul(AQUAPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (AquaFarm.sol#1536-1539)
        -accAQUAPerShare = accAQUAPerShare.add(AQUAReward.mul(1e12).div(sharesTotal)) (AquaFarm.sol#1540-1542)
PlanetFinance.updatePool(uint256) (AquaFarm.sol#1574-1599) performs a multiplication on the result of a division:
        -AQUAReward = multiplier.mul(AQUAPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (AquaFarm.sol#1588-1591)
        -pool.accAQUAPerShare = pool.accAQUAPerShare.add(AQUAReward.mul(1e12).div(sharesTotal)) (AquaFarm.sol#1595-1597)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Reentrancy in PlanetFinance.add(uint256,IERC20,bool,address) (AquaFarm.sol#1473-1494):
        External calls:
        - massUpdatePools() (AquaFarm.sol#1480)
                - AquaToken(AQUA).mint(address(this),AQUAReward) (AquaFarm.sol#1593)
        State variables written after the call(s):
        - poolInfo.push(PoolInfo(_want,_allocPoint,lastRewardBlock,0,_strat)) (AquaFarm.sol#1485-1493)
        - totalAllocPoint = totalAllocPoint.add(_allocPoint) (AquaFarm.sol#1484)
Reentrancy in PlanetFinance.deposit(uint256,uint256) (AquaFarm.sol#1602-1630):
        External calls:
        - updatePool(_pid) (AquaFarm.sol#1603)
                - AquaToken(AQUA).mint(address(this),AQUAReward) (AquaFarm.sol#1593)
        - safeAQUATransfer(msg.sender,pending) (AquaFarm.sol#1613)
                - IERC20(AQUA).transfer(_to,AQUABal) (AquaFarm.sol#1706)
                - IERC20(AQUA).transfer(_to,_AQUAAmt) (AquaFarm.sol#1708)
        - pool.want.safeTransferFrom(address(msg.sender),address(this),_wantAmt) (AquaFarm.sol#1617-1621)
        - pool.want.safeIncreaseAllowance(pool.strat,_wantAmt) (AquaFarm.sol#1623)
        - sharesAdded = IStrategy(poolInfo[_pid].strat).deposit(msg.sender,_wantAmt) (AquaFarm.sol#1624-1625)
        State variables written after the call(s):
        - user.shares = user.shares.add(sharesAdded) (AquaFarm.sol#1626)
        - user.rewardDebt = user.shares.mul(pool.accAQUAPerShare).div(1e12) (AquaFarm.sol#1628)
Reentrancy in PlanetFinance.emergencyWithdraw(uint256) (AquaFarm.sol#1685-1700):
        External calls:
        - IStrategy(poolInfo[_pid].strat).withdraw(msg.sender,amount) (AquaFarm.sol#1694)
        - pool.want.safeTransfer(address(msg.sender),amount) (AquaFarm.sol#1696)
        State variables written after the call(s):
        - user.shares = 0 (AquaFarm.sol#1698)
        - user.rewardDebt = 0 (AquaFarm.sol#1699)
Reentrancy in PlanetFinance.set(uint256,uint256,bool) (AquaFarm.sol#1497-1509):
        External calls:
        - massUpdatePools() (AquaFarm.sol#1503)
                - AquaToken(AQUA).mint(address(this),AQUAReward) (AquaFarm.sol#1593)
        State variables written after the call(s):
        - poolInfo[_pid].allocPoint = _allocPoint (AquaFarm.sol#1508)
        - totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint) (AquaFarm.sol#1505-1507)
Reentrancy in PlanetFinance.updatePool(uint256) (AquaFarm.sol#1574-1599):
        External calls:
        - AquaToken(AQUA).mint(address(this),AQUAReward) (AquaFarm.sol#1593)
        State variables written after the call(s):
        - pool.accAQUAPerShare = pool.accAQUAPerShare.add(AQUAReward.mul(1e12).div(sharesTotal)) (AquaFarm.sol#1595-1597)
        - pool.lastRewardBlock = block.number (AquaFarm.sol#1598)
Reentrancy in PlanetFinance.withdraw(uint256,uint256) (AquaFarm.sol#1633-1678):
        External calls:
        - updatePool(_pid) (AquaFarm.sol#1634)
                - AquaToken(AQUA).mint(address(this),AQUAReward) (AquaFarm.sol#1593)
        - safeAQUATransfer(msg.sender,pending) (AquaFarm.sol#1652)
                - IERC20(AQUA).transfer(_to,AQUABal) (AquaFarm.sol#1706)
                - IERC20(AQUA).transfer(_to,_AQUAAmt) (AquaFarm.sol#1708)
        - sharesRemoved = IStrategy(poolInfo[_pid].strat).withdraw(msg.sender,_wantAmt) (AquaFarm.sol#1661-1662)
        State variables written after the call(s):
        - user.shares = 0 (AquaFarm.sol#1665)
        - user.shares = user.shares.sub(sharesRemoved) (AquaFarm.sol#1667)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
PlanetFinance.emergencyWithdraw(uint256) (AquaFarm.sol#1685-1700) ignores return value by IStrategy(poolInfo[_pid].strat).withdraw(msg.sender,amount) (AquaFarm.sol#1694)
PlanetFinance.safeAQUATransfer(address,uint256) (AquaFarm.sol#1703-1710) ignores return value by IERC20(AQUA).transfer(_to,AQUABal) (AquaFarm.sol#1706)
PlanetFinance.safeAQUATransfer(address,uint256) (AquaFarm.sol#1703-1710) ignores return value by IERC20(AQUA).transfer(_to,_AQUAAmt) (AquaFarm.sol#1708)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

Re-entrancy vulnerabilities were not found by auditors. Safe functions are used in the code to prevent re-enntrancy attacks.

In addition, mathematical operation are well implemented.

AquaStrategy_4BELT.sol

```
INFO:Detectors:
StratX2.withdraw(address,uint256) (AquaStrategy_4BELT.sol#1858-1900) performs a multiplication on the result of a division:
        -_wantAmt = _wantAmt.mul(withdrawFeeFactor).div(withdrawFeeFactorMax) (AquaStrategy_4BELT.sol#1874-1876)
        -withdrawFee = _wantAmt.mul(withdrawFeeFactorMax.sub(withdrawFeeFactor)).div(withdrawFeeFactorMax) (AquaStrategy_4BELT.sol#1877)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Reentrancy in StratX2.deposit(address,uint256) (AquaStrategy_4BELT.sol#1790-1831):
        External calls:
        - IERC20(wantAddress).safeTransferFrom(address(msg.sender),address(this),_wantAmt) (AquaStrategy_4BELT.sol#1798-1802)
        - IERC20(wantAddress).safeIncreaseAllowance(depositFeeAddress,depositFee) (AquaStrategy_4BELT.sol#1820)
        - IERC20(wantAddress).transfer(depositFeeAddress,depositFee) (AquaStrategy_4BELT.sol#1821)
        State variables written after the call(s):
        - wantLockedTotal = wantLockedTotal.add(_wantAmt) (AquaStrategy_4BELT.sol#1827)
Reentrancy in StratX2.withdraw(address,uint256) (AquaStrategy_4BELT.sol#1858-1900):
        External calls:
        - IERC20(wantAddress).safeIncreaseAllowance(withdrawFeeAddress,withdrawFee) (AquaStrategy_4BELT.sol#1878)
        - IERC20(wantAddress).transfer(withdrawFeeAddress,withdrawFee) (AquaStrategy_4BELT.sol#1879)
        - _unfarm(_wantAmt) (AquaStrategy_4BELT.sol#1883)
                - IPancakeswapFarm(farmContractAddress).leaveStaking(_wantAmt) (AquaStrategy_4BELT.sol#1852)
                - IPancakeswapFarm(farmContractAddress).withdraw(pid,_wantAmt) (AquaStrategy_4BELT.sol#1854)
        State variables written after the call(s):
        - wantLockedTotal = wantLockedTotal.sub(_wantAmt) (AquaStrategy_4BELT.sol#1895)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
StratX2.deposit(address,uint256) (AquaStrategy_4BELT.sol#1790-1831) ignores return value by IERC20(wantAddress).transfer(depositFeeAddress,depositFee) (AquaStrategy_4BELT.sol#1821)
StratX2.withdraw(address,uint256) (AquaStrategy_4BELT.sol#1858-1900) ignores return value by IERC20(wantAddress).transfer(withdrawFeeAddress,withdrawFee) (AquaStrategy_4BELT.sol#1879)
StratX2.earn() (AquaStrategy_4BELT.sol#1906-1988) ignores return value by IPancakeRouter02(unlRouterAddress).addLiquidity(token0Address,token1Address,token0Amt,token1Amt,0,0,address(this),block.timestamp.add(600)) (AquaStrategy_4BELT.sol#1973-1982)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

Re-entrancy vulnerabilities were not found by auditors. Safe functions are used in the code to prevent re-enntrancy attacks.
In addition, mathematical operation are well implemented.

AquaStrategy_AQUA.sol

```
INFO:Detectors:
StratX2.withdraw(address,uint256) (AquaStrategy_AQUA.sol#1858-1900) performs a multiplication on the result of a division:
        -_wantAmt = _wantAmt.mul(withdrawFeeFactor).div(withdrawFeeFactorMax) (AquaStrategy_AQUA.sol#1874-1876)
        -withdrawFee = _wantAmt.mul(withdrawFeeFactorMax.sub(withdrawFeeFactor)).div(withdrawFeeFactorMax) (AquaStrategy_AQUA.sol#1877)
AquaStrategy_AQUA.withdraw(address,uint256) (AquaStrategy_AQUA.sol#2325-2373) performs a multiplication on the result of a division:
        -_wantAmt = _wantAmt.mul(withdrawFeeFactor).div(withdrawFeeFactorMax) (AquaStrategy_AQUA.sol#2347-2349)
        -withdrawFee = _wantAmt.mul(withdrawFeeFactorMax.sub(withdrawFeeFactor)).div(withdrawFeeFactorMax) (AquaStrategy_AQUA.sol#2350)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Reentrancy in StratX2.deposit(address,uint256) (AquaStrategy_AQUA.sol#1790-1831):
        External calls:
        - IERC20(wantAddress).safeTransferFrom(address(msg.sender),address(this),_wantAmt) (AquaStrategy_AQUA.sol#1798-1802)
        - IERC20(wantAddress).safeIncreaseAllowance(depositFeeAddress,depositFee) (AquaStrategy_AQUA.sol#1820)
        - IERC20(wantAddress).transfer(depositFeeAddress,depositFee) (AquaStrategy_AQUA.sol#1821)
        State variables written after the call(s):
        - wantLockedTotal = wantLockedTotal.add(_wantAmt) (AquaStrategy_AQUA.sol#1827)
Reentrancy in AquaStrategy_AQUA.deposit(address,uint256) (AquaStrategy_AQUA.sol#2280-2323):
        External calls:
        - IERC20(wantAddress).safeTransferFrom(address(msg.sender),address(this),_wantAmt) (AquaStrategy_AQUA.sol#2293-2297)
        - IERC20(wantAddress).safeIncreaseAllowance(depositFeeAddress,depositFee) (AquaStrategy_AQUA.sol#2316)
        - IERC20(wantAddress).transfer(depositFeeAddress,depositFee) (AquaStrategy_AQUA.sol#2317)
        State variables written after the call(s):
        - wantLockedTotal = IERC20(wantAddress).balanceOf(address(this)) (AquaStrategy_AQUA.sol#2320)
Reentrancy in StratX2.withdraw(address,uint256) (AquaStrategy_AQUA.sol#1858-1900):
        External calls:
        - IERC20(wantAddress).safeIncreaseAllowance(withdrawFeeAddress,withdrawFee) (AquaStrategy_AQUA.sol#1878)
        - IERC20(wantAddress).transfer(withdrawFeeAddress,withdrawFee) (AquaStrategy_AQUA.sol#1879)
        - _unfarm(_wantAmt) (AquaStrategy_AQUA.sol#1883)
                - IPancakeswapFarm(farmContractAddress).leaveStaking(_wantAmt) (AquaStrategy_AQUA.sol#1852)
                - IPancakeswapFarm(farmContractAddress).withdraw(pid,_wantAmt) (AquaStrategy_AQUA.sol#1854)
        State variables written after the call(s):
        - wantLockedTotal = wantLockedTotal.sub(_wantAmt) (AquaStrategy_AQUA.sol#1895)
Reentrancy in AquaStrategy_AQUA.withdraw(address,uint256) (AquaStrategy_AQUA.sol#2325-2373):
        External calls:
        - IERC20(wantAddress).safeIncreaseAllowance(withdrawFeeAddress,withdrawFee) (AquaStrategy_AQUA.sol#2351)
        - IERC20(wantAddress).transfer(withdrawFeeAddress,withdrawFee) (AquaStrategy_AQUA.sol#2352)
        State variables written after the call(s):
        - wantLockedTotal = wantLockedTotal.sub(_wantAmt) (AquaStrategy_AQUA.sol#2368)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
StratX2.deposit(address,uint256) (AquaStrategy_AQUA.sol#1790-1831) ignores return value by IERC20(wantAddress).transfer(depositFeeAddress,depositFee) (AquaStrategy_AQUA.sol#1821)
StratX2.withdraw(address,uint256) (AquaStrategy_AQUA.sol#1858-1900) ignores return value by IERC20(wantAddress).transfer(withdrawFeeAddress,withdrawFee) (AquaStrategy_AQUA.sol#1879)
StratX2.earn() (AquaStrategy_AQUA.sol#1906-1988) ignores return value by IPancakeRouter02(unlRouterAddress).addLiquidity(token0Address,token1Address,token0Amt,token1Amt,0,0,address(this),block.timestamp.add(600)) (AquaStrategy_AQUA.sol#1973-1982)
AquaStrategy_AQUA.deposit(address,uint256) (AquaStrategy_AQUA.sol#2280-2323) ignores return value by IERC20(wantAddress).transfer(depositFeeAddress,depositFee) (AquaStrategy_AQUA.sol#2317)
AquaStrategy_AQUA.withdraw(address,uint256) (AquaStrategy_AQUA.sol#2325-2373) ignores return value by IERC20(wantAddress).transfer(withdrawFeeAddress,withdrawFee) (AquaStrategy_AQUA.sol#2352)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

Re-entrancy vulnerabilities were not found by auditors. Safe functions are used in the code to prevent re-enntrancy attacks.
In addition, mathematical operation are well implemented.

AquaStrategy_PCS.sol

```
INFO:Detectors:
StratX2.withdraw(address,uint256) (AquaStrategy_PCS.sol#1858-1900) performs a multiplication on the result of a division:
        -_wantAmt = _wantAmt.mul(withdrawFeeFactor).div(withdrawFeeFactorMax) (AquaStrategy_PCS.sol#1874-1876)
        -withdrawFee = _wantAmt.mul(withdrawFeeFactorMax.sub(withdrawFeeFactor)).div(withdrawFeeFactorMax) (AquaStrategy_PCS.sol#1877)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Reentrancy in StratX2.deposit(address,uint256) (AquaStrategy_PCS.sol#1790-1831):
        External calls:
        - IERC20(wantAddress).safeTransferFrom(address(msg.sender),address(this),_wantAmt) (AquaStrategy_PCS.sol#1798-1802)
        - IERC20(wantAddress).safeIncreaseAllowance(depositFeeAddress,depositFee) (AquaStrategy_PCS.sol#1820)
        - IERC20(wantAddress).transfer(depositFeeAddress,depositFee) (AquaStrategy_PCS.sol#1821)
        State variables written after the call(s):
        - wantLockedTotal = wantLockedTotal.add(_wantAmt) (AquaStrategy_PCS.sol#1827)
Reentrancy in StratX2.withdraw(address,uint256) (AquaStrategy_PCS.sol#1858-1900):
        External calls:
        - IERC20(wantAddress).safeIncreaseAllowance(withdrawFeeAddress,withdrawFee) (AquaStrategy_PCS.sol#1878)
        - IERC20(wantAddress).transfer(withdrawFeeAddress,withdrawFee) (AquaStrategy_PCS.sol#1879)
        - _unfarm(_wantAmt) (AquaStrategy_PCS.sol#1883)
                - IPancakeswapFarm(farmContractAddress).leaveStaking(_wantAmt) (AquaStrategy_PCS.sol#1852)
                - IPancakeswapFarm(farmContractAddress).withdraw(pid,_wantAmt) (AquaStrategy_PCS.sol#1854)
        State variables written after the call(s):
        - wantLockedTotal = wantLockedTotal.sub(_wantAmt) (AquaStrategy_PCS.sol#1895)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
StratX2.deposit(address,uint256) (AquaStrategy_PCS.sol#1790-1831) ignores return value by IERC20(wantAddress).transfer(depositFeeAddress,depositFee) (AquaStrategy_PCS.sol#1821)
StratX2.withdraw(address,uint256) (AquaStrategy_PCS.sol#1858-1900) ignores return value by IERC20(wantAddress).transfer(withdrawFeeAddress,withdrawFee) (AquaStrategy_PCS.sol#1879)
StratX2.earn() (AquaStrategy_PCS.sol#1906-1988) ignores return value by IPancakeRouter02(unlRouterAddress).addLiquidity(token0Address,token1Address,token0Amt,token1Amt,0,0,address(this),block.timestamp.add(600)) (AquaStrategy_PCS.sol#1973-1982)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

Re-entrancy vulnerabilities were not found by auditors. Safe functions are used in the code to prevent re-enntrancy attacks.
In addition, mathematical operation are well implemented.

AquaToken.sol.

```
INFO:Detectors:
Pragma version^0.6.12 (AquaToken.sol#7) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
solc-0.6.12 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter AQUA.mint(address,uint256)._to (AquaToken.sol#657) is not in mixedCase
Parameter AQUA.mint(address,uint256)._amount (AquaToken.sol#657) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformity-to-solidity-naming-conventions
INFO:Detectors:
name() should be declared external:
        - ERC20.name() (AquaToken.sol#290-292)
symbol() should be declared external:
        - ERC20.symbol() (AquaToken.sol#298-300)
decimals() should be declared external:
        - ERC20.decimals() (AquaToken.sol#315-317)
totalSupply() should be declared external:
        - ERC20.totalSupply() (AquaToken.sol#322-324)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (AquaToken.sol#329-331)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (AquaToken.sol#341-349)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (AquaToken.sol#354-362)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (AquaToken.sol#371-379)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (AquaToken.sol#394-409)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (AquaToken.sol#423-434)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (AquaToken.sol#450-464)
owner() should be declared external:
        - Ownable.owner() (AquaToken.sol#617-619)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (AquaToken.sol#636-639)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (AquaToken.sol#645-652)
mint(address,uint256) should be declared external:
        - AQUA.mint(address,uint256) (AquaToken.sol#657-659)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:AquaToken.sol analyzed (6 contracts with 46 detectors), 19 result(s) found
```

The issue was identified by auditors in HAL11 – POSSIBLE MISUSE OF PUBLIC FUNCTIONS.

PlanetFactory.sol

```
INFO:Detectors:
PlanetPair._safeTransfer(address,address,uint256) (PlanetFactory.sol#293-296) uses a dangerous strict equality:
        - require(bool,string)(success && (data.length == 0 || abi.decode(data,(bool))),Planet: TRANSFER_FAILED) (PlanetFactory.sol#295)
PlanetPair.mint(address) (PlanetFactory.sol#359-380) uses a dangerous strict equality:
        - _totalSupply == 0 (PlanetFactory.sol#368)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in PlanetPair.burn(address) (PlanetFactory.sol#383-405):
        External calls:
        - _safeTransfer(_token0,to,amount0) (PlanetFactory.sol#397)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (PlanetFactory.sol#294)
        - _safeTransfer(_token1,to,amount1) (PlanetFactory.sol#398)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (PlanetFactory.sol#294)
        State variables written after the call(s):
        - _update(balance0,balance1,_reserve0,_reserve1) (PlanetFactory.sol#402)
                - blockTimestampLast = blockTimestamp (PlanetFactory.sol#333)
        - kLast = uint256(reserve0).mul(reserve1) (PlanetFactory.sol#403)
        - _update(balance0,balance1,_reserve0,_reserve1) (PlanetFactory.sol#402)
                - reserve0 = uint112(balance0) (PlanetFactory.sol#331)
        - _update(balance0,balance1,_reserve0,_reserve1) (PlanetFactory.sol#402)
                - reserve1 = uint112(balance1) (PlanetFactory.sol#332)
Reentrancy in PlanetFactory.createPair(address,address) (PlanetFactory.sol#471-486):
        External calls:
        - IPlanetPair(pair).initialize(token0,token1) (PlanetFactory.sol#481)
        State variables written after the call(s):
        - getPair[token0][token1] = pair (PlanetFactory.sol#482)
        - getPair[token1][token0] = pair (PlanetFactory.sol#483)
Reentrancy in PlanetPair.swap(uint256,uint256,address,bytes) (PlanetFactory.sol#408-436):
        External calls:
        - _safeTransfer(_token0,to,amount0Out) (PlanetFactory.sol#419)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (PlanetFactory.sol#294)
        - _safeTransfer(_token1,to,amount1Out) (PlanetFactory.sol#420)
                - (success,data) = token.call(abi.encodeWithSelector(SELECTOR,to,value)) (PlanetFactory.sol#294)
        - IPlanetCallee(to).planetCall(msg.sender,amount0Out,amount1Out,data) (PlanetFactory.sol#421)
        State variables written after the call(s):
        - _update(balance0,balance1,_reserve0,_reserve1) (PlanetFactory.sol#434)
                - blockTimestampLast = blockTimestamp (PlanetFactory.sol#333)
        - _update(balance0,balance1,_reserve0,_reserve1) (PlanetFactory.sol#434)
                - reserve0 = uint112(balance0) (PlanetFactory.sol#331)
        - _update(balance0,balance1,_reserve0,_reserve1) (PlanetFactory.sol#434)
                - reserve1 = uint112(balance1) (PlanetFactory.sol#332)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```

For the strict equality it won't cause any harm, especially when we do

not make the comparison with the balance since we can force the transfer of ether. Furthermore, re-entrancy vulnerabilities were not found by auditors. Safe functions are used in the code to prevent re-enntrancy attacks.

PlanetRouter.sol

```
INFO:Detectors:
PlanetLibrary.getAmountsOut(address,uint256,address[]).i (PlanetRouter.sol#341) is a local variable never initialized
PlanetRouter._swap(uint256[],address[],address).i (PlanetRouter.sol#594) is a local variable never initialized
PlanetRouter._swapSupportingFeeOnTransferTokens(address[],address).i (PlanetRouter.sol#703) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
PlanetRouter._addLiquidity(address,address,uint256,uint256,uint256,uint256) (PlanetRouter.sol#414-441) ignores return value by IPlanetFactory(factory).createPair(tokenA,tokenB) (PlanetRouter.sol#424)
PlanetRouter.removeLiquidity(address,address,uint256,uint256,uint256,address,uint256) (PlanetRouter.sol#484-500) ignores return value by IPlanetPair(pair).transferFrom(msg.sender,pair,liquidity) (PlanetRouter.sol#494)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

TimelockController.sol

```
INFO:Detectors:
Reentrancy in TimelockController.execute(address,uint256,bytes,bytes32,bytes32) (TimelockController.sol#1596-1607):
        External calls:
        - _call(id,0,target,value,data) (TimelockController.sol#1605)
                - (success) = target.call{value: value}(data) (TimelockController.sol#1677)
        State variables written after the call(s):
        - _afterCall(id) (TimelockController.sol#1606)
                - _timestamps[id] = _DONE_TIMESTAMP (TimelockController.sol#1661)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
TimelockController.withdrawBEP20(address) (TimelockController.sol#1788-1793) ignores return value by IERC20(_tokenAddress).transfer(devWalletAddress,tokenBal) (TimelockController.sol#1792)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
```

Re-entrancy vulnerabilities were not found by auditors. Safe functions are used in the code to prevent re-enntrancy attacks. In addition, low-level call was mentioned by auditors in HAL12 - USE OF LOW-LEVEL CALLS.

3.20 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. Only security-related findings were considered as in-scope.

Results:

AquaFarm.sol

Report for AquaFarm.sol
https://dashboard.mythx.io/#/console/analyses/2cdfcbde-d5d0-4f00-ae6c-ab766b2bd0ce

Line	SWC Title	Severity	Short Description
290	(SWC-000) Unknown	Medium	Function could be marked as external.
298	(SWC-000) Unknown	Medium	Function could be marked as external.
315	(SWC-000) Unknown	Medium	Function could be marked as external.
322	(SWC-000) Unknown	Medium	Function could be marked as external.
329	(SWC-000) Unknown	Medium	Function could be marked as external.
341	(SWC-000) Unknown	Medium	Function could be marked as external.
354	(SWC-000) Unknown	Medium	Function could be marked as external.
371	(SWC-000) Unknown	Medium	Function could be marked as external.
394	(SWC-000) Unknown	Medium	Function could be marked as external.
423	(SWC-000) Unknown	Medium	Function could be marked as external.
450	(SWC-000) Unknown	Medium	Function could be marked as external.
592	(SWC-131) Presence of unused variables	Low	Unused function parameter "from".
593	(SWC-131) Presence of unused variables	Low	Unused function parameter "to".
594	(SWC-131) Presence of unused variables	Low	Unused function parameter "amount".
744	(SWC-107) Reentrancy	Low	A call to a user-supplied address is executed.
744	(SWC-123) Requirement Violation	Low	Requirement violation.
1303	(SWC-000) Unknown	Medium	Function could be marked as external.
1322	(SWC-000) Unknown	Medium	Function could be marked as external.
1331	(SWC-000) Unknown	Medium	Function could be marked as external.
1387	(SWC-000) Unknown	Medium	Function could be marked as external.
1417	(SWC-123) Requirement Violation	Low	Requirement violation.
1473	(SWC-000) Unknown	Medium	Function could be marked as external.
1483	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1483	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	A control flow decision is made based on The block.number environment variable.
1497	(SWC-000) Unknown	Medium	Function could be marked as external.
1533	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1535	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1568	(SWC-128) DoS With Block Gas Limit	Medium	Loop over unbounded data structure.
1576	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1581	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1584	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1598	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1602	(SWC-000) Unknown	Medium	Function could be marked as external.
1680	(SWC-000) Unknown	Medium	Function could be marked as external.
1685	(SWC-000) Unknown	Medium	Function could be marked as external.
1712	(SWC-000) Unknown	Medium	Function could be marked as external.
1720	(SWC-000) Unknown	Medium	Function could be marked as external.

AquaStrategy_4BELT.sol

Report for AquaStrategy_4BELT.sol
https://dashboard.mythx.io/#/console/analyses/1fac7909-9f1e-4409-b91b-6b5227cb0ff2

Line	SWC Title	Severity	Short Description
294	(SWC-000) Unknown	Medium	Function could be marked as external.
302	(SWC-000) Unknown	Medium	Function could be marked as external.
319	(SWC-000) Unknown	Medium	Function could be marked as external.
326	(SWC-000) Unknown	Medium	Function could be marked as external.
333	(SWC-000) Unknown	Medium	Function could be marked as external.
345	(SWC-000) Unknown	Medium	Function could be marked as external.
358	(SWC-000) Unknown	Medium	Function could be marked as external.
375	(SWC-000) Unknown	Medium	Function could be marked as external.
398	(SWC-000) Unknown	Medium	Function could be marked as external.
427	(SWC-000) Unknown	Medium	Function could be marked as external.
454	(SWC-000) Unknown	Medium	Function could be marked as external.
596	(SWC-131) Presence of unused variables	Low	Unused function parameter "from".
597	(SWC-131) Presence of unused variables	Low	Unused function parameter "to".
598	(SWC-131) Presence of unused variables	Low	Unused function parameter "amount".
1307	(SWC-000) Unknown	Medium	Function could be marked as external.
1326	(SWC-000) Unknown	Medium	Function could be marked as external.
1652	(SWC-000) Unknown	Medium	Function could be marked as external.
1790	(SWC-131) Presence of unused variables	Low	Unused function parameter "_userAddress".
1790	(SWC-000) Unknown	Medium	Function could be marked as external.
1833	(SWC-000) Unknown	Medium	Function could be marked as external.
1858	(SWC-000) Unknown	Medium	Function could be marked as external.
1858	(SWC-131) Presence of unused variables	Low	Unused function parameter "_userAddress".
1906	(SWC-000) Unknown	Medium	Function could be marked as external.
1926	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1985	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
2036	(SWC-000) Unknown	Medium	Function could be marked as external.
2081	(SWC-000) Unknown	Medium	Function could be marked as external.
2085	(SWC-000) Unknown	Medium	Function could be marked as external.
2089	(SWC-000) Unknown	Medium	Function could be marked as external.
2137	(SWC-000) Unknown	Medium	Function could be marked as external.
2142	(SWC-000) Unknown	Medium	Function could be marked as external.
2147	(SWC-000) Unknown	Medium	Function could be marked as external.
2156	(SWC-000) Unknown	Medium	Function could be marked as external.
2165	(SWC-000) Unknown	Medium	Function could be marked as external.
2174	(SWC-000) Unknown	Medium	Function could be marked as external.
2192	(SWC-000) Unknown	Medium	Function could be marked as external.
2282	(SWC-000) Unknown	Medium	Function could be marked as external.
2302	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
2335	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
2340	(SWC-000) Unknown	Medium	Function could be marked as external.
2345	(SWC-000) Unknown	Medium	Function could be marked as external.

AquaStrategy_AQUA.sol

Report for AquaStrategy_AQUA.sol
https://dashboard.mythx.io/#/console/analyses/6135b63f-8acd-4e42-b932-356bfeef42c0

Line	SWC Title	Severity	Short Description
294	(SWC-000) Unknown	Medium	Function could be marked as external.
302	(SWC-000) Unknown	Medium	Function could be marked as external.
319	(SWC-000) Unknown	Medium	Function could be marked as external.
326	(SWC-000) Unknown	Medium	Function could be marked as external.
333	(SWC-000) Unknown	Medium	Function could be marked as external.
345	(SWC-000) Unknown	Medium	Function could be marked as external.
358	(SWC-000) Unknown	Medium	Function could be marked as external.
375	(SWC-000) Unknown	Medium	Function could be marked as external.
398	(SWC-000) Unknown	Medium	Function could be marked as external.
427	(SWC-000) Unknown	Medium	Function could be marked as external.
454	(SWC-000) Unknown	Medium	Function could be marked as external.
596	(SWC-131) Presence of unused variables	Low	Unused function parameter "from".
597	(SWC-131) Presence of unused variables	Low	Unused function parameter "to".
598	(SWC-131) Presence of unused variables	Low	Unused function parameter "amount".
1307	(SWC-000) Unknown	Medium	Function could be marked as external.
1326	(SWC-000) Unknown	Medium	Function could be marked as external.
1652	(SWC-000) Unknown	Medium	Function could be marked as external.
1790	(SWC-131) Presence of unused variables	Low	Unused function parameter "_userAddress".
1790	(SWC-000) Unknown	Medium	Function could be marked as external.
1833	(SWC-000) Unknown	Medium	Function could be marked as external.
1858	(SWC-000) Unknown	Medium	Function could be marked as external.
1858	(SWC-131) Presence of unused variables	Low	Unused function parameter "_userAddress".
1906	(SWC-000) Unknown	Medium	Function could be marked as external.
1926	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randomness.
1985	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randomness.
2036	(SWC-000) Unknown	Medium	Function could be marked as external.
2081	(SWC-000) Unknown	Medium	Function could be marked as external.
2085	(SWC-000) Unknown	Medium	Function could be marked as external.
2089	(SWC-000) Unknown	Medium	Function could be marked as external.
2137	(SWC-000) Unknown	Medium	Function could be marked as external.
2142	(SWC-000) Unknown	Medium	Function could be marked as external.
2147	(SWC-000) Unknown	Medium	Function could be marked as external.
2156	(SWC-000) Unknown	Medium	Function could be marked as external.
2165	(SWC-000) Unknown	Medium	Function could be marked as external.
2174	(SWC-000) Unknown	Medium	Function could be marked as external.
2192	(SWC-000) Unknown	Medium	Function could be marked as external.
2280	(SWC-000) Unknown	Medium	Function could be marked as external.
2325	(SWC-000) Unknown	Medium	Function could be marked as external.
2377	(SWC-131) Presence of unused variables	Low	Unused function parameter "_wantAmt".
2379	(SWC-000) Unknown	Medium	Function could be marked as external.
2406	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randomness.
2415	(SWC-000) Unknown	Medium	Function could be marked as external.
2424	(SWC-000) Unknown	Medium	Function could be marked as external.
2428	(SWC-000) Unknown	Medium	Function could be marked as external.
2433	(SWC-000) Unknown	Medium	Function could be marked as external.

AquaStrategy_PCS.sol

Report for AquaStrategy_PCS.sol
https://dashboard.mythx.io/#/console/analyses/c72c95e7-2c98-48da-8523-d139a2688a88

Line	SWC Title	Severity	Short Description
294	(SWC-000) Unknown	Medium	Function could be marked as external.
302	(SWC-000) Unknown	Medium	Function could be marked as external.
319	(SWC-000) Unknown	Medium	Function could be marked as external.
326	(SWC-000) Unknown	Medium	Function could be marked as external.
333	(SWC-000) Unknown	Medium	Function could be marked as external.
345	(SWC-000) Unknown	Medium	Function could be marked as external.
358	(SWC-000) Unknown	Medium	Function could be marked as external.
375	(SWC-000) Unknown	Medium	Function could be marked as external.
398	(SWC-000) Unknown	Medium	Function could be marked as external.
427	(SWC-000) Unknown	Medium	Function could be marked as external.
454	(SWC-000) Unknown	Medium	Function could be marked as external.
596	(SWC-131) Presence of unused variables	Low	Unused function parameter "from".
597	(SWC-131) Presence of unused variables	Low	Unused function parameter "to".
598	(SWC-131) Presence of unused variables	Low	Unused function parameter "amount".
1307	(SWC-000) Unknown	Medium	Function could be marked as external.
1326	(SWC-000) Unknown	Medium	Function could be marked as external.
1652	(SWC-000) Unknown	Medium	Function could be marked as external.
1790	(SWC-000) Unknown	Medium	Function could be marked as external.
1790	(SWC-131) Presence of unused variables	Low	Unused function parameter "_userAddress".
1833	(SWC-000) Unknown	Medium	Function could be marked as external.
1858	(SWC-131) Presence of unused variables	Low	Unused function parameter "_userAddress".
1858	(SWC-000) Unknown	Medium	Function could be marked as external.
1906	(SWC-000) Unknown	Medium	Function could be marked as external.
1926	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1985	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
2036	(SWC-000) Unknown	Medium	Function could be marked as external.
2081	(SWC-000) Unknown	Medium	Function could be marked as external.
2085	(SWC-000) Unknown	Medium	Function could be marked as external.
2089	(SWC-000) Unknown	Medium	Function could be marked as external.
2137	(SWC-000) Unknown	Medium	Function could be marked as external.
2142	(SWC-000) Unknown	Medium	Function could be marked as external.
2147	(SWC-000) Unknown	Medium	Function could be marked as external.
2156	(SWC-000) Unknown	Medium	Function could be marked as external.
2165	(SWC-000) Unknown	Medium	Function could be marked as external.
2174	(SWC-000) Unknown	Medium	Function could be marked as external.
2192	(SWC-000) Unknown	Medium	Function could be marked as external.
2272	(SWC-000) Unknown	Medium	Function could be marked as external.
2277	(SWC-000) Unknown	Medium	Function could be marked as external.

AquaToken.sol.

Report for AquaToken.sol
https://dashboard.mythx.io/#/console/analyses/39bc0694-314b-451e-a4f1-245c0f86279f

Line	SWC Title	Severity	Short Description
7	(SWC-103) Floating Pragma	Low	A floating pragma is set.
290	(SWC-000) Unknown	Medium	Function could be marked as external.
298	(SWC-000) Unknown	Medium	Function could be marked as external.
315	(SWC-000) Unknown	Medium	Function could be marked as external.
322	(SWC-000) Unknown	Medium	Function could be marked as external.
329	(SWC-000) Unknown	Medium	Function could be marked as external.
341	(SWC-000) Unknown	Medium	Function could be marked as external.
354	(SWC-000) Unknown	Medium	Function could be marked as external.
371	(SWC-000) Unknown	Medium	Function could be marked as external.
394	(SWC-000) Unknown	Medium	Function could be marked as external.
423	(SWC-000) Unknown	Medium	Function could be marked as external.
450	(SWC-000) Unknown	Medium	Function could be marked as external.
590	(SWC-131) Presence of unused variables	Low	Unused function parameter "from".
591	(SWC-131) Presence of unused variables	Low	Unused function parameter "to".
592	(SWC-131) Presence of unused variables	Low	Unused function parameter "amount".
617	(SWC-000) Unknown	Medium	Function could be marked as external.
636	(SWC-000) Unknown	Medium	Function could be marked as external.
645	(SWC-000) Unknown	Medium	Function could be marked as external.
657	(SWC-000) Unknown	Medium	Function could be marked as external.

PlanetFactory.sol

No issues were found.

TimelockController.sol

Report for TimelockController.sol
https://dashboard.mythx.io/#/console/analyses/0c840a2e-afa2-466b-99f4-1b20af5f31a2

Line	SWC Title	Severity	Short Description
1301	(SWC-123) Requirement Violation	Low	Requirement violation.
1437	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
1728	(SWC-101) Integer Overflow and Underflow	High	The arithmetic operation can overflow.
1841	(SWC-123) Requirement Violation	Low	Requirement violation.

THANK YOU FOR CHOOSING

// HALBORN