// HALBORN

Tidal Finance

Smart Contracts Security Audit

Prepared by: **Halborn**

Date of Engagement: **June 25, 2021 - June 29, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	06/25/2021	Gabi Urrutia
0.3	Document Edits	06/26/2021	Gokberk Gulgun
1.0	Final Draft	06/29/2021	Gabi Urrutia
1.1	Remediation Plan	07/06/2021	Gabi Urrutia
1.1	Remediation Review	07/06/2021	Steve Walbroehl

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
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Tidal Finance engaged Halborn to conduct a security assessment on their Smart contract beginning on June 25th, 2021 and ending June 29th, 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 Tidal Finance prior to its production release shortly following the assessments deadline.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure smart-contract development.

1.2 AUDIT SUMMARY

The team at Halborn was provided a week 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 few 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 IDE)

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

All contracts included in the Tidal Finance Repository

Commit ID: 7012a90e9adb6472af067925014aeb40a50fea36.
Fix Commit ID: 149d95251e0591ca1487a9d0a1a18fb6784fc577

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	5	6

LIKELIHOOD

IMPACT

	(HAL-02) (HAL-04) (HAL-05)			
	(HAL-01) (HAL-03)			
(HAL-06) (HAL-07) (HAL-08) (HAL-09) (HAL-10) (HAL-11)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - EXPERIMENTAL FEATURES ENABLED	Low	RISK ACCEPTED
HAL02 - MISSING ACCESS CONTROL ON THE TRUSTED FORWARDER FUNCTION	Low	SOLVED - 07/02/2021
HAL03 - MISSING EVENT HANDLER	Low	RISK ACCEPTED
HAL04 - FLOATING PRAGMA	Low	SOLVED - 07/02/2021
HAL05 - OWNER CAN RENOUNCE OWNERSHIP	Low	ACKNOWLEDGED
HAL06 - INFINITE MINTING	Informational	NOT APPLICABLE
HAL07 - USE OF INLINE ASSEMBLY	Informational	RISK ACCEPTED
HAL08 - MISSING RE-ENTRANCY PROTECTION	Informational	SOLVED - 07/02/2021
HAL09 - BLOCK TIMESTAMP ALIAS USAGE	Informational	NOT APPLICABLE
HAL10 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED - 07/02/2021
HAL11 - MISSING CONSTANT DEFINITION	Informational	SOLVED - 07/02/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) 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-bug/

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 exploitation. The bug, when triggered, will under certain circumstances send corrupt parameters on method invocations to other contracts.

Code Location:

GovernorAlpha.sol Line #14

Listing 1: GovernorAlpha.sol (Lines 15)

```
14 pragma solidity 0.6.12;
15 pragma experimental ABIEncoderV2;
```

Staking.sol Line #3

Listing 2: Staking.sol (Lines 3)

```
3 pragma solidity 0.6.12;
4 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: Tidal.Finance accepts the use of `ABIEncoderV2` in `view` functions and for parsing structured data.

3.2 (HAL-02) MISSING ACCESS CONTROL ON THE TRUSTED FORWARDER FUNCTION - LOW

Description:

During the tests, It has been observed that, only owner check is missing on the setTrustedForwarder function.

Code Location:

Registry.sol Line #123

Listing 3: Registry.sol (Lines 123)

```
123     function setTrustedForwarder(address trustedForwarder_)
            external {
124         trustedForwarder = trustedForwarder_;
125     }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It is recommended to implement access control in that function.

Remediation Plan:

SOLVED: Tidal.Finance implemented access control in setTrustedForwarder function in commit b7dae61079b1dddd913b819ace516b9f0fd83a92.

3.3 (HAL-03) MISSING EVENT HANDLER - LOW

Description:

In the Registry.sol contract the function does 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.

Code Location:

Registry.sol Line #~46-125

Listing 4: Registry.sol (Lines)

```
46     function setTimeExtra(uint256 timeExtra_)
47     function setBuyer(address buyer_)
48     function setSeller(address seller_)
49     function setGuarantor(address guarantor_)
50     function setStaking(address staking_)
51     function setBonus(address bonus_)
52     function setTidalToken(address tidalToken_)
53     function setBaseToken(address baseToken_)
54     function setAssetManager(address assetManager_)
55     function setPremiumCalculator(address premiumCalculator_)
56     function setPlatform(address platform_)
57     function setGuarantorPercentage(uint256 percentage_)
58     function setPlatformPercentage(uint256 percentage_)
59     function setGovernor(address governor_)
60     function setCommittee(address committee_)
61     function setTrustedForwarder(address trustedForwarder_)
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

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

Remediation Plan:

RISK ACCEPTED: Tidal.Finance considers it to be appropriate not declaring these events because most of these functions will be called only once.

3.4 (HAL-04) FLOATING PRAGMA - LOW

Description:

In the BaseRelayRecipient.sol,the contract uses the floating pragma ^0.6.12. 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:

BaseRelayRecipient Line #2

Listing 5: BaseRelayRecipient.sol (Lines 1)

```
1 pragma solidity ^0.6.12;
```

- This is an example where the floating pragma is used. ^0.6.12.

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Consider locking the pragma version. It is recommended not to use floating pragma in production. Apart from just locking the pragma version in the code, the sign (>=) need to be removed. It is possible to lock the pragma version both in truffle-config.js and in hardhat.config.js if you use HardHat framework for the deployment.

Remediation Plan:

SOLVED: Tidal.Finance locked the pragma version (0.6.12) in commit 6ba4efc80292769ebd8a36d3ade87b7ac5bb6ea0.

3.5 (HAL-05) OWNER CAN RENOUNCE OWNERSHIP - LOW

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 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. All contracts are affected which is derived from Ownable contract.

Code Location:

PremiumCalculator.sol Line #11
Bonus.sol Line #18
AssetManager.sol Line #9
Registry.sol Line #8
CommitteeAlpha.sol Line #18
Buyer.sol Line #22
Staking.sol Line #18

Risk Level:

Likelihood - 2
Impact - 3

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.

Listing 6: Disable RenounceOwnerShip (Lines 1)

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

Remediation Plan:

ACKNOWLEDGED: Tidal.Finance will transfer Owners to Timelock and multi-signature wallet or DAO will indirectly control it.

3.6 (HAL-06) INFINITE MINTING - INFORMATIONAL

Description:

During the test, it has been observed that an attacker could mint any amount of USDC. Although, the comment is written on the top of the code, the code should be deleted from the repository.

Code Location:

MockUSDC.sol Line #3

Listing 7: MockUSDC.sol (Lines 45)

```
44      // For test purpose.
45      function mint(uint256 amount_) external {
46          _mint(msg.sender, amount_);
47      }
```

Recommendation:

It is recommend to delete related code from the repository.

Remediation Plan:

NOT APPLICABLE: Tidal.Finance claims that MockUSDC.sol file is for testing purposes only and will never be used in production.

3.7 (HAL-07) 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:

GovernanceToken.sol Line #248
GovernorAlpha.sol Line #39
StakingHelper.sol Line #20

Listing 8: GovernanceToken.sol (Lines)

```
248     constructor() public {
249         uint chainId;
250         assembly {
251             chainId := chainid
252         }
```

Listing 9: BaseRelayRecipient.sol (Lines)

```
35 function _msgSender() internal override virtual view returns (
       address payable ret) {
36         if (msg.data.length >= 24 && isTrustedForwarder(msg.sender
               )) {
37             // At this point we know that the sender is a trusted
                   forwarder,
38             // so we trust that the last bytes of msg.data are the
                    verified sender address.
39             // extract sender address from the end of msg.data
40             assembly {
41                 ret := shr(96,calldataload(sub(calldatasize(),20))
                       )
42             }
43         } else {
44             return msg.sender;
```

```
45          }
46 }
```

Risk Level:

Likelihood - 1
Impact - 1

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: Tidal.Finance assumes the risk because the use of assembly is needed.

3.8 (HAL-08) MISSING RE-ENTRANCY PROTECTION - INFORMATIONAL

Description:

To protect against cross-function reentrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the withdraw function with a recursive call. OpenZeppelin has it's own mutex implementation called ReentrancyGuard which provides a modifier to any function called nonReentrant that guards the function with a mutex against reentrancy attacks.

Code Location:

Staking.sol Line #259

Listing 10: Staking.sol (Lines 259)

```
259     function claim() external {
260         UserInfo storage user = userInfo[_msgSender()];
261
262         updatePool();
263
264         uint256 pending = user.amount.mul(poolInfo.
                accRewardPerShare).div(
265             UNIT_PER_SHARE).sub(user.rewardDebt);
266         uint256 rewardTotal = user.rewardAmount.add(pending);
267
268         IERC20(registry.tidalToken()).transfer(_msgSender(),
                rewardTotal);
269
270         user.rewardAmount = 0;
271         user.rewardDebt = user.amount.mul(poolInfo.
                accRewardPerShare).div(UNIT_PER_SHARE);
272
273         emit Claim(_msgSender(), rewardTotal);
274     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

In the Staking.sol , the claim() function is missing nonReentrant guard. Use the nonReentrant or mutex modifier to avoid introducing future vulnerabilities.

Remediation Plan:

SOLVED: Tidal.Finance added Re-entrancy protection in commit 7984f69a292180c9393b4eedeeef40ef5217d2e1.

3.9 (HAL-09) BLOCK TIMESTAMP ALIAS USAGE - INFORMATIONAL

Description:

The global variable now alias of the block.timestamp does not necessarily hold the current time, and may not be accurate. Miners can influence the value of now 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:

WeekManaged.sol Line #11,15

Listing 11: WeekManaged.sol (Lines 11,15)

```
4  abstract contract WeekManaged {
5
6      uint256 public offset = 4 days;
7
8      function _timeExtra() internal virtual view returns(uint256);
9
10     function getCurrentWeek() public view returns(uint256) {
11         return (now + offset + _timeExtra()) / (7 days);
12     }
13
14     function getNow() public view returns(uint256) {
15         return now + _timeExtra();
16     }
17
18     function getUnlockWeek() public view returns(uint256) {
19         return getCurrentWeek() + 2;
20     }
21
22     function getUnlockTime(uint256 time_) public view returns(
           uint256) {
23         require(time_ + offset > (7 days), "Time not large enough"
               );
```

```
24          return ((time_ + offset) / (7 days) + 2) * (7 days) -
                offset;
25      }
26  }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Use block.number instead of 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:

NOT APPLICABLE: Tidal.Finance considers it appropriate to use block.timestamp for their business logic. In addition, the timescale used in the project is higher than 900 seconds.

3.10 (HAL-10) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In 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:

Buyer.sol Line #106

Listing 12: Buyer.sol (Lines)

```
106     function getBalance(address who_) public view returns(uint256)
            {
107         return userInfoMap[who_].balance;
108     }
```

Buyer.sol Line #97

Listing 13: Buyer.sol (Lines)

```
 97     function getPremiumRate(uint16 assetIndex_) public view
           returns(uint256) {
 98         return IPremiumCalculator(registry.premiumCalculator()).
               getPremiumRate(
 99             assetIndex_);
100     }
```

Staking.sol Line #276

Listing 14: Staking.sol (Lines)

```
276     function isAssetLocked(address who_) public view returns(bool)
            {
277         return payoutId > 0 && !payoutInfo[payoutId].finished &&
               userPayoutIdMap[who_] < payoutId;
278     }
```

Risk Level:

Likelihood - 1
Impact - 1

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:

SOLVED: Tidal.Finance declared external instead of public functions in commit 149d95251e0591ca1487a9d0a1a18fb6784fc577.

3.11 (HAL-11) MISSING CONSTANT DEFINITION - INFORMATIONAL

Description:

State variables should be declared constant to save gas. Without constant definition, the state variable reading progress is performed through the SLOAD operation which costs 200 gas alone.

Code Location:

Staking.sol Line #64

Listing 15: Staking.sol (Lines 64)

```
64     uint256 public withdrawWaitTime = 14 days;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Add the constant attributes to state variables that never change.

Remediation Plan:

SOLVED: Tidal.Finance added constant attributes to the state variable in commit e9eba5c6066718501a6e842c73f21a49ab6041a6.

3.12 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, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire codebase.

Results:

```
INFO:Detectors:
Reentrancy in Bonus.updateGuarantorBonus(uint16) (contracts/Bonus.sol#54-62):
        External calls:
        - IERC20(registry.tidalToken()).approve(registry.guarantor(),bonusPerAssetOfG[assetIndex_]) (contracts/Bonus.sol#58)
        - IGuarantor(registry.guarantor()).updateBonus(assetIndex_,bonusPerAssetOfG[assetIndex_]) (contracts/Bonus.sol#59)
        State variables written after the call(s):
        - guarantorWeek[assetIndex_] = currentWeek (contracts/Bonus.sol#61)
Reentrancy in Bonus.updateSellerBonus(uint16) (contracts/Bonus.sol#44-52):
        External calls:
        - IERC20(registry.tidalToken()).approve(registry.seller(),bonusPerAssetOfS[assetIndex_]) (contracts/Bonus.sol#48)
        - ISeller(registry.seller()).updateBonus(assetIndex_,bonusPerAssetOfS[assetIndex_]) (contracts/Bonus.sol#49)
        State variables written after the call(s):
        - sellerWeek[assetIndex_] = currentWeek (contracts/Bonus.sol#51)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
Bonus.updateSellerBonus(uint16) (contracts/Bonus.sol#44-52) ignores return value by IERC20(registry.tidalToken()).approve(registry.seller(),bonusPerAssetOfS[assetIndex_]) (contracts/Bonus.sol
#48)
Bonus.updateGuarantorBonus(uint16) (contracts/Bonus.sol#54-62) ignores return value by IERC20(registry.tidalToken()).approve(registry.guarantor(),bonusPerAssetOfG[assetIndex_]) (contracts/Bon
us.sol#58)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
Bonus.updateSellerBonus(uint16) (contracts/Bonus.sol#44-52) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(sellerWeek[assetIndex_] < currentWeek,Already updated) (contracts/Bonus.sol#46)
Bonus.updateGuarantorBonus(uint16) (contracts/Bonus.sol#54-62) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(guarantorWeek[assetIndex_] < currentWeek,Already updated) (contracts/Bonus.sol#56)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Pragma version0.6.12 (contracts/Bonus.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/WeekManaged.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/interfaces/IBonus.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/interfaces/IBuyer.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/interfaces/IGuarantor.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/interfaces/IRegistry.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/interfaces/ISeller.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (node_modules/@openzeppelin/contracts/math/SafeMath.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (node_modules/@openzeppelin/contracts/utils/Context.sol#3) 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:
Function IRegistry.PERCENTAGE_BASE() (contracts/interfaces/IRegistry.sol#6) is not in mixedCase
Function IRegistry.UTILIZATION_BASE() (contracts/interfaces/IRegistry.sol#7) is not in mixedCase
Function IRegistry.PREMIUM_BASE() (contracts/interfaces/IRegistry.sol#8) is not in mixedCase
Function IRegistry.UNIT_PER_SHARE() (contracts/interfaces/IRegistry.sol#9) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformity-to-solidity-naming-conventions
```

```
INFO:Detectors:
getNow() should be declared external:
        - WeekManaged.getNow() (contracts/WeekManaged.sol#16-18)
getUnlockWeek() should be declared external:
        - WeekManaged.getUnlockWeek() (contracts/WeekManaged.sol#20-22)
getUnlockTime(uint256) should be declared external:
        - WeekManaged.getUnlockTime(uint256) (contracts/WeekManaged.sol#24-27)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#54-57)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#63-67)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Detectors:
Pragma version0.6.12 (contracts/PremiumCalculator.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/interfaces/IAssetManager.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/interfaces/IBuyer.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/interfaces/IPremiumCalculator.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/interfaces/IRegistry.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (node_modules/@openzeppelin/contracts/utils/Context.sol#3) 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:
Function IRegistry.PERCENTAGE_BASE() (contracts/interfaces/IRegistry.sol#6) is not in mixedCase
Function IRegistry.UTILIZATION_BASE() (contracts/interfaces/IRegistry.sol#7) is not in mixedCase
Function IRegistry.PREMIUM_BASE() (contracts/interfaces/IRegistry.sol#8) is not in mixedCase
Function IRegistry.UNIT_PER_SHARE() (contracts/interfaces/IRegistry.sol#9) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformity-to-solidity-naming-conventions
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#54-57)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#63-67)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Detectors:
AssetManager.setAsset(uint16,address,uint8).asset (contracts/AssetManager.sol#32) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
Pragma version0.6.12 (contracts/AssetManager.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/interfaces/IAssetManager.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (node_modules/@openzeppelin/contracts/utils/Context.sol#3) 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:
Variable Migrations.last_completed_migration (contracts/Migrations.sol#6) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformity-to-solidity-naming-conventions
INFO:Detectors:
setCompleted(uint256) should be declared external:
        - Migrations.setCompleted(uint256) (contracts/Migrations.sol#16-18)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Detectors:
Seller.changeBasket(uint8,uint16[]) (contracts/Seller.sol#162-193) uses a dangerous strict equality:
        - require(bool,string)(userInfo[msg.sender].week == getCurrentWeek(),Not updated yet) (contracts/Seller.sol#164)
Seller.changeBasketReady(address,uint8) (contracts/Seller.sol#195-226) uses a dangerous strict equality:
        - require(bool,string)(userInfo[who_].week == getCurrentWeek(),Not updated yet) (contracts/Seller.sol#199)
Seller.deposit(uint8,uint256) (contracts/Seller.sol#286-293) uses a dangerous strict equality:
        - require(bool,string)(userInfo[msg.sender].week == getCurrentWeek(),Not updated yet) (contracts/Seller.sol#288)
Seller.reduceDeposit(uint8,uint256) (contracts/Seller.sol#295-305) uses a dangerous strict equality:
        - require(bool,string)(userInfo[msg.sender].week == getCurrentWeek(),Not updated yet) (contracts/Seller.sol#298)
Seller.update(address) (contracts/Seller.sol#229-284) uses a dangerous strict equality:
        - require(bool,string)(poolInfo[index].weekOfPremium == week && poolInfo[index].weekOfBonus == week,Not ready) (contracts/Seller.sol#240-241)
Seller.updatePremium(uint16) (contracts/Seller.sol#99-113) uses a dangerous strict equality:
        - require(bool,string)(IBuyer(registry.buyer()).weekToUpdate() == week,buyer not ready) (contracts/Seller.sol#101)
Seller.withdraw(uint8,uint256) (contracts/Seller.sol#307-319) uses a dangerous strict equality:
        - require(bool,string)(userInfo[msg.sender].week == getCurrentWeek(),Not updated yet) (contracts/Seller.sol#309)
Seller.withdrawReady(address,uint8) (contracts/Seller.sol#321-350) uses a dangerous strict equality:
        - require(bool,string)(userInfo[who_].week == getCurrentWeek(),Not updated yet) (contracts/Seller.sol#325)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in Seller.claimBonus() (contracts/Seller.sol#357-360):
        External calls:
        - IERC20(registry.tidalToken()).safeTransfer(msg.sender,userInfo[msg.sender].bonus) (contracts/Seller.sol#358)
        State variables written after the call(s):
        - userInfo[msg.sender].bonus = 0 (contracts/Seller.sol#359)
Reentrancy in Seller.claimPremium() (contracts/Seller.sol#352-355):
        External calls:
        - IERC20(registry.baseToken()).safeTransfer(msg.sender,userInfo[msg.sender].premium) (contracts/Seller.sol#353)
        State variables written after the call(s):
        - userInfo[msg.sender].premium = 0 (contracts/Seller.sol#354)
Reentrancy in Seller.finishPayout(uint16,uint256) (contracts/Seller.sol#414-431):
        External calls:
        - IERC20(registry.baseToken()).safeTransferFrom(msg.sender,address(this),payoutInfo[assetIndex_][payoutId_].total.sub(payoutInfo[assetIndex_][payoutId_].paid)) (contracts/Seller.sol#4
20-423)
        State variables written after the call(s):
        - payoutInfo[assetIndex_][payoutId_].paid = payoutInfo[assetIndex_][payoutId_].total (contracts/Seller.sol#424)
Reentrancy in Seller.finishPayout(uint16,uint256) (contracts/Seller.sol#414-431):
        External calls:
        - IERC20(registry.baseToken()).safeTransferFrom(msg.sender,address(this),payoutInfo[assetIndex_][payoutId_].total.sub(payoutInfo[assetIndex_][payoutId_].paid)) (contracts/Seller.sol#4
20-423)
        - IERC20(registry.baseToken()).safeTransfer(payoutInfo[assetIndex_][payoutId_].toAddress,payoutInfo[assetIndex_][payoutId_].total) (contracts/Seller.sol#427-428)
        State variables written after the call(s):
        - payoutInfo[assetIndex_][payoutId_].finished = true (contracts/Seller.sol#430)
```

```
INFO:Detectors:
Seller.changeBasket(uint8,uint16[]).request (contracts/Seller.sol#185) is a local variable never initialized
Seller.withdraw(uint8,uint256).request (contracts/Seller.sol#314) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
Seller.isAssetLocked(address,uint8) (contracts/Seller.sol#132-142) has external calls inside a loop: i < IAssetManager(registry.assetManager()).getIndexesByCategoryLength(category_) (contract
s/Seller.sol#133)
Seller.isAssetLocked(address,uint8) (contracts/Seller.sol#132-142) has external calls inside a loop: index = IAssetManager(registry.assetManager()).getIndexesByCategory(category_,i) (contract
s/Seller.sol#134)
Seller.changeBasket(uint8,uint16[]) (contracts/Seller.sol#162-193) has external calls inside a loop: i < IAssetManager(registry.assetManager()).getIndexesByCategoryLength(category_) (contract
s/Seller.sol#171)
Seller.changeBasket(uint8,uint16[]) (contracts/Seller.sol#162-193) has external calls inside a loop: index = uint16(IAssetManager(registry.assetManager()).getIndexesByCategory(category_,i)) (
contracts/Seller.sol#173)
Seller.changeBasketReady(address,uint8) (contracts/Seller.sol#195-226) has external calls inside a loop: i < IAssetManager(registry.assetManager()).getIndexesByCategoryLength(category_) (cont
racts/Seller.sol#209)
Seller.changeBasketReady(address,uint8) (contracts/Seller.sol#195-226) has external calls inside a loop: index = uint16(IAssetManager(registry.assetManager()).getIndexesByCategory(category_,i
)) (contracts/Seller.sol#211)
Seller.update(address) (contracts/Seller.sol#229-284) has external calls inside a loop: index < IAssetManager(registry.assetManager()).getAssetLength() (contracts/Seller.sol#238)
Seller.update(address) (contracts/Seller.sol#229-284) has external calls inside a loop: index < IAssetManager(registry.assetManager()).getAssetLength() (contracts/Seller.sol#248)
Seller.update(address) (contracts/Seller.sol#229-284) has external calls inside a loop: category = IAssetManager(registry.assetManager()).getAssetCategory(index) (contracts/Seller.sol#250)
Seller.update(address) (contracts/Seller.sol#229-284) has external calls inside a loop: userInfo[who_].premium = userInfo[who_].premium.add(currentBalance.mul(poolInfo[index].premiumPerShare)
.div(registry.UNIT_PER_SHARE())) (contracts/Seller.sol#255-256)
Seller.update(address) (contracts/Seller.sol#229-284) has external calls inside a loop: userInfo[who_].bonus = userInfo[who_].bonus.add(currentBalance.mul(poolInfo[index].bonusPerShare).div(r
egistry.UNIT_PER_SHARE())) (contracts/Seller.sol#259-260)
Seller.update(address) (contracts/Seller.sol#229-284) has external calls inside a loop: category < IAssetManager(registry.assetManager()).getCategoryLength() (contracts/Seller.sol#270)
Seller.withdrawReady(address,uint8) (contracts/Seller.sol#321-350) has external calls inside a loop: i < IAssetManager(registry.assetManager()).getIndexesByCategoryLength(category_) (contract
s/Seller.sol#335)
Seller.withdrawReady(address,uint8) (contracts/Seller.sol#321-350) has external calls inside a loop: index = IAssetManager(registry.assetManager()).getIndexesByCategory(category_,i) (contract
s/Seller.sol#337)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in Seller.deposit(uint8,uint256) (contracts/Seller.sol#286-293):
        External calls:
        - IERC20(registry.baseToken()).safeTransferFrom(msg.sender,address(this),amount_) (contracts/Seller.sol#290)
        State variables written after the call(s):
        - userBalance[msg.sender][category_].futureBalance = userBalance[msg.sender][category_].futureBalance.add(amount_) (contracts/Seller.sol#292)
Reentrancy in Seller.withdrawReady(address,uint8) (contracts/Seller.sol#321-350):
        External calls:
        - IERC20(registry.baseToken()).safeTransfer(who_,request.amount) (contracts/Seller.sol#332)
        State variables written after the call(s):
        - assetBalance[index] = assetBalance[index].sub(request.amount) (contracts/Seller.sol#341)
        - categoryBalance[category_] = categoryBalance[category_].sub(request.amount) (contracts/Seller.sol#347)
        - userBalance[who_][category_].currentBalance = userBalance[who_][category_].currentBalance.sub(request.amount) (contracts/Seller.sol#345)
        - userBalance[who_][category_].futureBalance = userBalance[who_][category_].futureBalance.sub(request.amount) (contracts/Seller.sol#346)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

```
INFO:Detectors:
Seller.updatePremium(uint16) (contracts/Seller.sol#99-113) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(IBuyer(registry.buyer()).weekToUpdate() == week,buyer not ready) (contracts/Seller.sol#101)
        - require(bool,string)(poolInfo[assetIndex_].weekOfPremium < week,already updated) (contracts/Seller.sol#102)
Seller.updateBonus(uint16,uint256) (contracts/Seller.sol#116-130) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(poolInfo[assetIndex_].weekOfBonus < week,already updated) (contracts/Seller.sol#121)
Seller.changeBasket(uint8,uint16[]) (contracts/Seller.sol#162-193) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(userInfo[msg.sender].week == getCurrentWeek(),Not updated yet) (contracts/Seller.sol#164)
Seller.changeBasketReady(address,uint8) (contracts/Seller.sol#195-226) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(userInfo[who_].week == getCurrentWeek(),Not updated yet) (contracts/Seller.sol#199)
        - require(bool,string)(getNow() > unlockTime,Not ready to change yet) (contracts/Seller.sol#204)
Seller.update(address) (contracts/Seller.sol#229-284) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(userInfo[who_].week < week,Already updated) (contracts/Seller.sol#233)
        - require(bool,string)(poolInfo[index].weekOfPremium == week && poolInfo[index].weekOfBonus == week,Not ready) (contracts/Seller.sol#240-241)
Seller.deposit(uint8,uint256) (contracts/Seller.sol#286-293) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(userInfo[msg.sender].week == getCurrentWeek(),Not updated yet) (contracts/Seller.sol#288)
Seller.reduceDeposit(uint8,uint256) (contracts/Seller.sol#295-305) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(userInfo[msg.sender].week == getCurrentWeek(),Not updated yet) (contracts/Seller.sol#298)
Seller.withdraw(uint8,uint256) (contracts/Seller.sol#307-319) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(userInfo[msg.sender].week == getCurrentWeek(),Not updated yet) (contracts/Seller.sol#309)
Seller.withdrawReady(address,uint8) (contracts/Seller.sol#321-350) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(userInfo[who_].week == getCurrentWeek(),Not updated yet) (contracts/Seller.sol#325)
        - require(bool,string)(getNow() > unlockTime,Not ready to withdraw yet) (contracts/Seller.sol#330)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (node_modules/@openzeppelin/contracts/utils/Address.sol#26-35) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#33)
Address._verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#171-188) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#180-183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Pragma version0.6.12 (contracts/NonReentrancy.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/Seller.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/WeekManaged.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/interfaces/IAssetManager.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/interfaces/IBuyer.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/interfaces/IRegistry.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (contracts/interfaces/ISeller.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (node_modules/@openzeppelin/contracts/math/SafeMath.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (node_modules/@openzeppelin/contracts/token/ERC20/SafeERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (node_modules/@openzeppelin/contracts/utils/Address.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (node_modules/@openzeppelin/contracts/utils/Context.sol#3) 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:
setDelay(uint256) should be declared external:
        - Timelock.setDelay(uint256) (contracts/Timelock.sol#38-45)
acceptAdmin() should be declared external:
        - Timelock.acceptAdmin() (contracts/Timelock.sol#47-53)
setPendingAdmin(address) should be declared external:
        - Timelock.setPendingAdmin(address) (contracts/Timelock.sol#55-66)
queueTransaction(address,uint256,string,bytes,uint256) should be declared external:
        - Timelock.queueTransaction(address,uint256,string,bytes,uint256) (contracts/Timelock.sol#68-77)
cancelTransaction(address,uint256,string,bytes,uint256) should be declared external:
        - Timelock.cancelTransaction(address,uint256,string,bytes,uint256) (contracts/Timelock.sol#79-86)
executeTransaction(address,uint256,string,bytes,uint256) should be declared external:
        - Timelock.executeTransaction(address,uint256,string,bytes,uint256) (contracts/Timelock.sol#88-113)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Detectors:
GovernorAlpha.execute(uint256) (contracts/GovernorAlpha.sol#226-234) sends eth to arbitrary user
        Dangerous calls:
        - timelock.executeTransaction.value(proposal.values[i])(proposal.targets[i],proposal.values[i],proposal.signatures[i],proposal.calldatas[i],proposal.eta) (contracts/GovernorAlpha.sol#
231)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
GovernanceToken._writeCheckpoint(address,uint32,uint256,uint256) (contracts/tokens/GovernanceToken.sol#223-241) uses a dangerous strict equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (contracts/tokens/GovernanceToken.sol#233)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
GovernorAlpha._queueOrRevert(address,uint256,string,bytes,uint256) (contracts/GovernorAlpha.sol#221-224) ignores return value by timelock.queueTransaction(target,value,signature,data,eta) (co
ntracts/GovernorAlpha.sol#223)
GovernorAlpha.execute(uint256) (contracts/GovernorAlpha.sol#226-234) ignores return value by timelock.executeTransaction.value(proposal.values[i])(proposal.targets[i],proposal.values[i],propo
sal.signatures[i],proposal.calldatas[i],proposal.eta) (contracts/GovernorAlpha.sol#231)
GovernorAlpha.__queueSetTimelockPendingAdmin(address,uint256) (contracts/GovernorAlpha.sol#325-328) ignores return value by timelock.queueTransaction(address(timelock),0,setPendingAdmin(addre
ss),abi.encode(newPendingAdmin),eta) (contracts/GovernorAlpha.sol#327)
GovernorAlpha.__executeSetTimelockPendingAdmin(address,uint256) (contracts/GovernorAlpha.sol#330-333) ignores return value by timelock.executeTransaction(address(timelock),0,setPendingAdmin(a
ddress),abi.encode(newPendingAdmin),eta) (contracts/GovernorAlpha.sol#332)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
GovernorAlpha.cancel(uint256).state (contracts/GovernorAlpha.sol#237) shadows:
        - GovernorAlpha.state(uint256) (contracts/GovernorAlpha.sol#260-280) (function)
GovernanceToken.constructor(string,string)._name (contracts/tokens/GovernanceToken.sol#9) shadows:
        - ERC20._name (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#42) (state variable)
GovernanceToken.constructor(string,string)._symbol (contracts/tokens/GovernanceToken.sol#9) shadows:
        - ERC20._symbol (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#43) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
GovernorAlpha.execute(uint256) (contracts/GovernorAlpha.sol#226-234) has external calls inside a loop: timelock.executeTransaction.value(proposal.values[i])(proposal.targets[i],proposal.value
s[i],proposal.signatures[i],proposal.calldatas[i],proposal.eta) (contracts/GovernorAlpha.sol#231)
GovernorAlpha.cancel(uint256) (contracts/GovernorAlpha.sol#236-249) has external calls inside a loop: timelock.cancelTransaction(proposal.targets[i],proposal.values[i],proposal.signatures[i],
proposal.calldatas[i],proposal.eta) (contracts/GovernorAlpha.sol#245)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
```

According to the test results, most of the findings found by slither were considered as false positives. Relevant findings were reviewed by the auditors.

3.13 AUTOMATED SECURITY SCAN RESULTS

Description:

Halborn used automated security scanners to assist with detection of well known security issues, and identify low-hanging fruit on the scoped contract targeted 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 MythX to locate any vulnerabilities. Security Detections are only in scope, and the analysis was pointed towards issues with the in-scope smart contracts.

Results:

AssetManager.sol

```
Report for contracts/AssetManager.sol
https://dashboard.mythx.io/#/console/analyses/8cb6f896-fa2c-4a89-82e4-7fb123803375
```

Line	SWC Title	Severity	Short Description
45	(SWC-128) DoS With Block Gas Limit	Medium	Implicit loop over unbounded data structure.
47	(SWC-128) DoS With Block Gas Limit	Medium	Loop over unbounded data structure.

Bonus.sol

```
Report for contracts/Bonus.sol
https://dashboard.mythx.io/#/console/analyses/f9ebb201-261a-4eca-9357-394ffbd36a3e
```

Line	SWC Title	Severity	Short Description
58	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.

Buyer.sol

```
Report for contracts/Buyer.sol
https://dashboard.mythx.io/#/console/analyses/d9e86e1b-6457-4344-84cf-104fb0320f83
```

Line	SWC Title	Severity	Short Description
102	(SWC-000) Unknown	Medium	Function could be marked as external.
106	(SWC-000) Unknown	Medium	Function could be marked as external.

CommitteeAlpha.sol

No issues were found by MythX.

Guarantor.sol

Report for contracts/Guarantor.sol
https://dashboard.mythx.io/#/console/analyses/185f3945-cef8-42a2-bf28-fa4bf59f6883

Line	SWC Title	Severity	Short Description
80	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.

PremiumCalculator.sol

No issues were found by MythX.

Registry.sol

No issues were found by MythX.

Seller.sol

Report for contracts/Seller.sol
https://dashboard.mythx.io/#/console/analyses/211c4f09-0b8f-4589-947f-762bf9adfff3

Line	SWC Title	Severity	Short Description
92	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.

Staking.sol

Report for contracts/Staking.sol
https://dashboard.mythx.io/#/console/analyses/bb5df9b5-c35e-4548-ac20-aa2060f6a9d1

Line	SWC Title	Severity	Short Description
82	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
146	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
147	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
159	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
165	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
169	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
175	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
276	(SWC-000) Unknown	Medium	Function could be marked as external.
318	(SWC-128) DoS With Block Gas Limit	Medium	Loop over unbounded data structure.
359	(SWC-128) DoS With Block Gas Limit	Low	Loop over unbounded data structure.

Timelock.sol

Report for contracts/Timelock.sol
https://dashboard.mythx.io/#/console/analyses/ff2fc181-ca83-44ea-92db-041242a75b85

Line	SWC Title	Severity	Short Description
38	(SWC-000) Unknown	Medium	Function could be marked as external.
47	(SWC-000) Unknown	Medium	Function could be marked as external.
55	(SWC-000) Unknown	Medium	Function could be marked as external.
68	(SWC-000) Unknown	Medium	Function could be marked as external.
79	(SWC-000) Unknown	Medium	Function could be marked as external.
88	(SWC-000) Unknown	Medium	Function could be marked as external.
103	(SWC-128) DoS With Block Gas Limit	Low	Potentially unbounded data structure passed to builtin.

tokens/GovernanceToken.sol

Report for contracts/tokens/GovernanceToken.sol
https://dashboard.mythx.io/#/console/analyses/7c31e824-94a2-4694-88ec-633a9d1f1a29

Line	SWC Title	Severity	Short Description
102	(SWC-128) DoS With Block Gas Limit	Low	Potentially unbounded data structure passed to builtin.
158	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
177	(SWC-128) DoS With Block Gas Limit	Low	Loop over unbounded data structure.
231	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

tokens/TidalToken.sol

No issues were found by MythX.

helper/StakingHelper.sol

No issues were found by MythX.

helper/UpdateHelper.sol

Report for helper/UpdateHelper.sol
https://dashboard.mythx.io/#/console/analyses/d83df1f3-c67f-4700-b7dd-15719b03a37e

Line	SWC Title	Severity	Short Description
17	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
18	(SWC-107) Reentrancy	Low	Read of persistent state following external call
18	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.

All relevant findings were founded in the manual code review.

THANK YOU FOR CHOOSING

// HALBORN