HALBORN

Seascape - ZombieFarm

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **December 26th, 2021 - January 21st, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	01/17/2022	Ferran Celades
0.2	Document Edits	01/18/2022	Ferran Celades
0.3	Draft Review	01/18/2022	Gabi Urrutia
1.0	Remediation Plan	01/24/2022	Ferran Celades
1.1	Remediation Plan Review	01/25/2022	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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Seascape engaged Halborn to conduct a security audit on their smart contracts beginning on December 26th, 2021 and ending on January 21st, 2022. The security assessment was scoped to the smart contracts provided in the GitHub repository seascape-smartcontracts/tree/game5-auditFix

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert 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 partially addressed and acknowledged by the Seascape team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding 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 bridge code 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 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
- 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 to the following smart contracts:

- defi/Stake.sol
- defi/StakeNft.sol
- defi/StakeToken.sol
- game_5/ZombieFarm.sol
- game_5/challenges/NftTokenChallenge.sol
- game_5/challenges/ScapeNftComboChallenge.sol
- game_5/challenges/SingleNftChallenge.sol
- game_5/challenges/SingleTokenChallenge.sol
- game_5/helpers/VaultHandler.sol
- game_5/interfaces/ZombieFarmChallengeInterface.sol
- game_5/interfaces/ZombieFarmInterface.sol
- game_5/interfaces/ZombieFarmRewardInterface.sol
- game_5/rewards/ScapeNftReward.sol
- All contracts inherited by these contracts

Commit ID: b8f8e6598a4d8591188b562f9bfa29a85bd7b71b

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	2	5	1	8

LIKELIHOOD

IMPACT

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

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 MISSING TOKEN DECIMALS CHECK	High	SOLVED - 01/20/2022
HAL-02 MISSING SUPPORT FOR DEFLATIONARY TOKENS	High	RISK ACCEPTED
HAL-03 CREATED SESSIONS CAN OVERLAP	Medium	SOLVED - 01/20/2022
HAL-04 INVALID PERIOD CHECK	Medium	SOLVED - 01/20/2022
HAL-05 INVALID REWARD AMOUNT	Medium	SOLVED - 01/20/2022
HAL-06 REWARD NFT RE-USAGE	Medium	NOT APPLICABLE
HAL-07 SESSION DUPLICATION	Medium	SOLVED - 01/20/2022
HAL-08 UNUSED AND UNVERIFIED VARIABLE	Low	RISK ACCEPTED
HAL-09 INVALID REPORTED VALUE	Informational	ACKNOWLEDGED
HAL-10 MISSING ZERO CHECKS	Informational	ACKNOWLEDGED
HAL-11 REWARD AND CHALLENGES CAN BE ADDED DURING SESSION	Informational	ACKNOWLEDGED
HAL-12 FEATURE NOT IMPLEMENTED	Informational	ACKNOWLEDGED
HAL-13 SESSION ID OF 0 ALLOWED	Informational	ACKNOWLEDGED
HAL-14 FUNCTION VISIBILITY RESTRICTION IN SCAPENFTREWARD	Informational	ACKNOWLEDGED
HAL-15 FUNCTION VISIBILITY RESTRICTION IN STAKENFT AND STAKETOKEN	Informational	ACKNOWLEDGED
HAL-16 DEPRECATED METHODS	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) MISSING TOKEN DECIMALS CHECK - HIGH

Description:

The newPeriod function under the StakeToken contract and the constructor of the SingleTokenChallenge and NftTokenChallenge contracts do not check for their decimals being the same as the SCALED factor. If a token whose decimals are != 18 will cause the staking period calculations to be wrong, including the rewards.

Code Location:

Listing 1: contracts/defi/StakeToken.sol (Lines 43,44)

```
29      function newPeriod(
30          uint256 key,
31          address stakeToken,
32          address rewardToken,
33          uint256 startTime,
34          uint256 endTime,
35          uint256 rewardPool
36      )
37          external
38      {
39          newStakePeriod(key, startTime, endTime, rewardPool);
40
41          // Challenge.stake is not null, means that earn is not
                null too.
42          Period storage period       = periods[msg.sender][key];
43          period.stakeToken           = stakeToken;
44          period.rewardToken          = rewardToken;
45
46          emit NewPeriod(msg.sender, key, stakeToken, rewardToken,
                startTime, endTime);
47      }
```

Listing 2: contracts/game_5/challenges/SingleTokenChallenge.sol (Lines 78,79)

```
74      constructor (address _zombieFarm, address _vault, address
            _stake, address _reward, address _stakeHandler)
            VaultHandler(_vault) public {
75          require(_zombieFarm != address(0), "invalid _zombieFarm
                address");
76
77          zombieFarm          = _zombieFarm;
78          stakeToken          = _stake;
79          rewardToken         = _reward;
80          stakeHandler        = _stakeHandler;
81
82          initReentrancyStatus();
83      }
```

Listing 3: contracts/game_5/challenges/NftTokenChallenges.sol (Lines 91)

```
84      constructor (address _zombieFarm, address _vault, address _nft
            , address _stake, address _reward, address _stakeHandler)
            VaultHandler(_vault) public {
85          require(_zombieFarm != address(0), "invalid _zombieFarm
                address");
86          require(_nft      != address(0), "data.stake verification
                failed");
87
88          zombieFarm          = _zombieFarm;
89          stakeToken          = _stake;
90          nft                 = _nft;
91          rewardToken         = _reward;
92          stakeHandler        = _stakeHandler;
93
94          initReentrancyStatus();
95      }
```

Risk Level:

Likelihood - 3

Impact - 5

Recommendation:

The code on `newPeriod` creation must check for both `stakeToken` and `rewardToken` decimals to be 18 the same as `SCALED` under `Stake` contract.

Remediation Plan:

SOLVED: The code now checks for the token decimals on all the aforementioned contracts.

3.2 (HAL-02) MISSING SUPPORT FOR DEFLATIONARY TOKENS - HIGH

Description:

The transferFromVaultToUser function under the VaultHandler contract does not check the difference between the transferred before/after balance. The transferFromUserToVault does perform this check, but the return value is never checked. Not checking those values could cause the rewards to have discrepancy with the actual staked amount of tokens.

Code Location:

Listing 4: contracts/defi/StakeToken.sol (Lines 29,33,46)

```
21      function transferFromUserToVault(address token, uint256 amount
            , address user) internal returns(uint256) {
22          if (token == address(0)) {
23              require(msg.value >= amount, "VAULT_HANDLER: not
                    enough native token");
24              return msg.value;
25          }
26          IERC20 _token = IERC20(token);
27          require(_token.balanceOf(user) >= amount, "VAULT_HANDLER:
                user has not enough token");
28
29          uint256 preTotalAmount = _token.balanceOf(vault);
30
31          _token.safeTransferFrom(user, vault, amount);
32
33          uint256 actualAmount = _token.balanceOf(vault) -
                preTotalAmount;
34
35          return actualAmount;
36      }
37
38      function transferFromVaultToUser(address token, uint256 amount
            , address user) internal returns(uint256) {
39          if (token == address(0)) {
40              payable(user).transfer(amount);
```

```
41              return amount;
42          }
43          IERC20 _token = IERC20(token);
44          require(_token.balanceOf(vault) >= amount, "VAULT_HANDLER:
                vault has not enough token");
45
46          _token.safeTransferFrom(vault, user, amount);
47
48          return amount;
49      }
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

The code should check for the real deposited amount and not rely on the function call parameter. If the changes are not made, the Seascape team should make sure that the provided tokens for staking/reward are not deflationary.

Remediation Plan:

RISK ACCEPTED: The SeaScape team accepts the risk on this finding.

3.3 (HAL-03) CREATED SESSIONS CAN OVERLAP - MEDIUM

Description:

The `startSession` function under the `ZombieFarm` contract does not check for the previous session id end time and allows creating a session whose start time is in the middle of the previous session period.

Risk Level:

Likelihood - 2
Impact - 5

Recommendation:

The code should check that the created session is not inside the previous session by checking the `startTime` of the new session to be bigger than the last session `endTime`.

Remediation Plan:

SOLVED: The code now checks for `isStarting` during session start, which will prevent a new session to be started if a previous was already started (even if not active).

3.4 (HAL-04) INVALID PERIOD CHECK - MEDIUM

Description:

The getPeriodTime function does always report endTime if the startTime is in the future. This can cause all reward and claims to be treated as if the period was finished.
If getPeriodTime is used inside a function that does not use the whenStakePeriodActive modifier, it will be vulnerable. Function using getPeriodTime are:

- claimable: Since the staker deposit is only updated when inside the startTIme/endTime range, it will return 0 before even reaching the invalid getPeriodTime
- updatePeriodClaimable (internal): Called using the updateRewardClaimable modifier. It is used on the deposit function after the whenStakePeriodActive. Also used on the withdraw function that will fail since no deposit was made yet. It is also called on reward function, but deposit is checked there for being not zero.
- _reward (internal): The code does check staker.deposit == 0 which will only change its value during deposit. The deposit function does call whenStakePeriodActive.

Code Location:

Listing 5: contracts/defi/Stake.sol (Lines 267,278)

```
265     function getPeriodTime(uint startTime, uint endTime) internal
            view returns(uint) {
266         if (!isActive(startTime, endTime)) {
267             return endTime;
268         }
269
270         return block.timestamp;
271     }
272
```

```
273     function isActive(uint startTime, uint endTime) internal view
            returns(bool) {
274         if (startTime == 0) {
275             return false;
276         }
277
278         return (block.timestamp >= startTime && block.timestamp <=
                endTime);
279     }
```

Risk Level:

Likelihood - 2
Impact - 5

Recommendation:

The `getPeriodTime` should not return `endTime` if the period hasn't started yet. All functions calling this method should make sure that the returned value is inside the period.

Remediation Plan:

SOLVED: The `getPeriodTime` function now checks for `block.timestamp < startTime` and reports `startTime` instead of `endTime`.

3.5 (HAL-05) INVALID REWARD AMOUNT - MEDIUM

Description:

If the reward is less than the difference between the start and end timestamps, the unit (reward per second) will always be 0. As an example, if the reward was 10000 and the timestamp diff was 1641929035 - 1441929035 (200000000) the unit value would be 0 causing all the reward calculations to not increase. When the unit value is used it gets factored against the SCALED variable, this means that the unit value itself should be stored SCALED already and not rely on lazy escalation.

```
>>> s.newStakePeriod(0, 0, int(time.time()), int(time.time()))
Transaction sent: 0x104bd477156db60ddb50023bde45b2379b304ad817c8cbf95262081e2a1a44bb
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  Stake.newStakePeriod confirmed - Block: 10   Gas used: 37612 (0.31%)

<Transaction '0x104bd477156db60ddb50023bde45b2379b304ad817c8cbf95262081e2a1a44bb'>
>>> s.stakePeriods(a[0], 0)
(0, 1641929030, 1641929030, 1, 0, 0, 0, 0)
>>> s.newStakePeriod(0, 0, int(time.time()), int(time.time()-1))
Transaction sent: 0xac04cbe465aa0b9a528da3763530f5acfffe0781037e53302976eb50ed12583c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 10
  Stake.newStakePeriod confirmed - Block: 11   Gas used: 26812 (0.22%)

<Transaction '0xac04cbe465aa0b9a528da3763530f5acfffe0781037e53302976eb50ed12583c'>
>>> s.stakePeriods(a[0], 0)
(0, 1641929035, 1641929034, 0, 0, 0, 0, 0)
>>>
```

Code Location:

Listing 6: contracts/defi/Stake.sol (Lines 85)

```
71      function newStakePeriod(
72          uint key,               // a unique identifier. could be a
                session id.
73          uint startTime,
74          uint endTime,
75          uint rewardPool
76      )
77          internal
78          validStakePeriodParams(key, startTime, endTime, rewardPool
               )
```

```
79      {
80          // Challenge.stake is not null, means that earn is not
                null too.
81          StakePeriod storage period       = stakePeriods[msg.sender
                ][key];
82          period.rewardPool           = rewardPool;
83          period.startTime            = startTime;
84          period.endTime              = endTime;
85          period.unit                 = rewardPool / (endTime -
                startTime);
86          period.rewardClaimableTime  = startTime;
87
88          emit NewStakePeriod(msg.sender, key, startTime, endTime);
89      }
```

Risk Level:

Likelihood - 2
Impact - 5

Recommendations:

It is recommended that the stored period.unit value is scaled by SCALER factor during declaration and not when used.

Remediation Plan:

SOLVED: The code now stores the unit value scaled with the SCALER factor.

3.6 (HAL-06) REWARD NFT RE-USAGE - MEDIUM

Description:

The addLevelRewardToSession under the ZombieFarm does allow setting the same _data.imgId to multiple levelId's including the Grand reward. This will deny the prize to be collected since it was already collected on a previous level.

```
>>>
>>> farm.addLevelRewardToSession(1, 1, reward, _data)
Transaction sent: 0x73b207dba7e4bca0f8d6ce8b91b0cc2bd864b5d73511eab7478bfd4074ba647e
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 178
  ZombieFarm.addLevelRewardToSession confirmed - Block: 179   Gas used: 141902 (1.18%)

<Transaction '0x73b207dba7e4bca0f8d6ce8b91b0cc2bd864b5d73511eab7478bfd4074ba647e'>
>>> farm.addLevelRewardToSession(1, 2, reward, _data)
Transaction sent: 0xb943af561f3fc398c61cc6ce530f375d7de350f6a0789f9f6b31c4f035b4592f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 179
  ZombieFarm.addLevelRewardToSession confirmed - Block: 180   Gas used: 141902 (1.18%)

<Transaction '0xb943af561f3fc398c61cc6ce530f375d7de350f6a0789f9f6b31c4f035b4592f'>
>>>
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendations:

The code should check for ID duplication on the _data parameters of the AddLevelToSession function under ScapeNftReward contract.

Remediation Plan:

NOT APPLICABLE: The team states that: The imgId passed as a reward is a metadata ticker. It is not an identifier of NFT. When the reward is claimed, the ZombieFarm will call a factory by passing metadata, so that factory will mint a new NFT and generate a new id for that NFT.

3.7 (HAL-07) SESSION DUPLICATION - MEDIUM

Description:

The value `startTime = 0` is allowed under the `validStakePeriodParams` modifier, bypassing the existence check and creating a duplicated session ID.

Code Location:

Listing 7: contracts/defi/Stake.sol (Lines 45)

```
45     modifier validStakePeriodParams(uint key, uint startTime, uint
           endTime, uint rewardPool) {
46         require(startTime < endTime,                              "
               STAKE_TOKEN: invalid_time");
47         require(rewardPool > 0,                                   "
               STAKE_TOKEN: zero_value");
48         require(stakePeriods[msg.sender][key].startTime == 0,     "
               STAKE_TOKEN: period_exists");
49         _;
50     }
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

Although the check is performed on the call stack parent, it is recommended to perform all check on the call stack leaf so extreme scenarios are not missed. The `validStakePeriodParams` modifier should either check for `startTime != 0` or use `endTime` for the `period_exists` check.

Remediation Plan:

SOLVED: The check was added to the `validStakePeriodParams` modifier. The modifier now enforces that `startTime >= block.timestamp`.

3.8 (HAL-08) UNUSED AND UNVERIFIED VARIABLE - LOW

Description:

In the ZombieFarm contract inside the startSession function, the grand reward variable is not used and not verified for being a valid reward. Since this value cannot be changed, this can lead to the grand reward not claimable.

Code Location:

Listing 8: contracts/game_5/ZombieFarm.sol (Lines 166,169)

```
151     function startSession(
152         uint256          startTime,
153         uint256          period,
154         uint8            levelAmount,
155         uint256          speedUpFee,
156         uint256          repickFee,
157         address          grandReward
158     )
159         external
160         onlyOwner
161     {
162         //
163         // Verifying the Grand reward
164         //
165         require(supportedRewards[grandReward], "unsupported reward
                ");
166         ZombieFarmRewardInterface reward =
                ZombieFarmRewardInterface(grandReward);
167
168         // Check that Grand Reward is valid: the rewardData and
                reward id should be parsable.
169         // require(reward.isValidData(rewardData), "Invalid reward
                 data");
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to validate the given address and to verify that the given contract does conform to the casted interface.

Remediation Plan:

RISK ACCEPTED: The SeaScape team accepts the risk on this finding.

3.9 (HAL-09) INVALID REPORTED VALUE - INFORMATIONAL

Description:

The initiated function does report true even if the session is in the future and hasn't started yet:

```
>>>
>>> s.newStakePeriod(1, now() + 1000, now() + 2000 , 1234)
Transaction sent: 0xf158b5e746e8e1367efb27be2ea6d82c382021747adaff5decaf90f680080ab6
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  Stake.newStakePeriod confirmed - Block: 5   Gas used: 125203 (1.04%)

<Transaction '0xf158b5e746e8e1367efb27be2ea6d82c382021747adaff5decaf90f680080ab6'>
>>> s.initiated(a[0], 1)
True
>>>
```

Code Location:

Listing 9: contracts/defi/Stake.sol (Lines 298)

```
294     function initiated(address namespace, uint key) public view
            returns(bool) {
295         if (key == 0) return false;
296
297         StakePeriod storage period = stakePeriods[namespace][key];
298         return (block.timestamp <= period.endTime);
299     }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

The function should check for the period.startTime and report true only if inside the startTime/endTime period.

Remediation Plan:

ACKNOWLEDGED: The SeaScape team acknowledged this finding.

3.10 (HAL-10) MISSING ZERO CHECKS - INFORMATIONAL

Description:

The addChallengeToSession under the SingleTokenChallenge contract should check for levelId != 0

Code Location:

Listing 10: contracts/game_5/ZombieFarm.sol (Lines 506)

```
502     function isStarting(uint8 sessionId) internal view returns(
            bool) {
503         if (sessionId == 0) {
504             return false;
505         }
506         return (now <= sessions[sessionId].startTime + sessions[
                sessionId].period);
507     }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to validate all argument values, including the default values of 0.

Remediation Plan:

ACKNOWLEDGED: The SeaScape team acknowledged this finding.

3.11 (HAL-11) REWARD AND CHALLENGES CAN BE ADDED DURING SESSION - INFORMATIONAL

Description:

The isStarting function under ZombieFarm does not skip active sessions. It will report true for sessions that are already started but not finished. This allows addChallengeToSession and addLevelRewardToSession functions to be called even after the session started.

Code Location:

Listing 11: contracts/game_5/ZombieFarm.sol (Lines 506)

```
502     function isStarting(uint8 sessionId) internal view returns(
            bool) {
503         if (sessionId == 0) {
504             return false;
505         }
506         return (now <= sessions[sessionId].startTime + sessions[
                sessionId].period);
507     }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

RISK ACCEPTED: The isStarting function is used in parallel with isActive to verify a session.

Remediation Plan:

ACKNOWLEDGED: The SeaScape team acknowledged this finding.

3.12 (HAL-12) FEATURE NOT IMPLEMENTED - INFORMATIONAL

Description:

The repick feature under the ZombieFarm contract states that a random challenge should be picked. However, the code does not perform any setting action to the playerChallenges mapping with a valid sessionChallenges and only removes the fee amount from the player.

Code Location:

Listing 12: contracts/game_5/ZombieFarm.sol (Lines 289)

```
279     function repick(uint256 sessionId, uint8 slotId, address
            challenge) external {
280         require(slotId >= 0 && slotId < 3, "invalid slot id");
281         require(sessionId > 0, "sessionId or challengeId is 0");
282         require(isActive(sessionId));
283         require(sessionChallenges[sessionId][challenge], "!session
                .challenge");
284
285         ZombieFarmChallengeInterface zombieChallenge =
                ZombieFarmChallengeInterface(challenge);
286         uint8 levelId = zombieChallenge.getLevel(sessionId);
287         require(levelId > 0, "no challenge");
288
289         require(playerChallenges[sessionId][levelId][msg.sender][
                slotId] == address(0), "already staked");
290
291         uint256 fee = sessions[sessionId].repickFee;
292
293         require(crowns.spendFrom(msg.sender, fee), "failed to
                spend fee");
294
295         emit Repick(sessionId, levelId, slotId, challenge, msg.
                sender, fee);
296     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

If the feature is not fully implemented, it is recommended to perform a revert rather than having a partial implementation and have this information documented.

Recommendation:

ACKNOWLEDGED: The SeaScape team acknowledged this finding.

3.13 (HAL-13) SESSION ID OF 0 ALLOWED - INFORMATIONAL

Description:

The newStakePeriod under the Stake.sol contract period function does allow adding a stake period for session ID of 0, which will always return none active when calling the isActive function.

Code Location:

Listing 13: contracts/defi/Stake.sol (Lines 45)

```
45     modifier validStakePeriodParams(uint key, uint startTime, uint
           endTime, uint rewardPool) {
46         require(startTime < endTime,                              "
               STAKE_TOKEN: invalid_time");
47         require(rewardPool > 0,                                   "
               STAKE_TOKEN: zero_value");
48         require(stakePeriods[msg.sender][key].startTime == 0,     "
               STAKE_TOKEN: period_exists");
49         _;
50     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Although the check is performed on the call stack parent, it is recommended to perform all check on the call stack leaf so extreme scenarios are not missed. The validStakePeriodParams modifier should check for key != 0.

Recommendation:

ACKNOWLEDGED: The SeaScape team acknowledged this finding.

3.14 (HAL-14) FUNCTION VISIBILITY RESTRICTION IN SCAPENFTREWARD - INFORMATIONAL

Description:

Function visibility on the `decode` function under the `ScapeNftReward` contract could be restricted to `pure` since the function does not read from state.

Code Location:

Listing 14: contracts/game_5/ScapeNftReward.sol (Lines 95)

```
94      function decode(bytes calldata data)
95          external view returns(uint256, uint256, uint8, address,
                uint256)
96      {
97          (
98              uint256 i,
99              uint256 j,
100             uint8 x,
101             address y,
102             uint256 z
103         ) = abi.decode(data, (uint256, uint256, uint8, address,
                uint256));
104
105         return (i, j, x, y, z);
106     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to change the function visibility to pure instead of view.

Recommendation:

ACKNOWLEDGED: The SeaScape team acknowledged this finding.

3.15 (HAL-15) FUNCTION VISIBILITY RESTRICTION IN STAKENFT AND STAKETOKEN - INFORMATIONAL

Description:

When creating a new period on either StakeNFT or StakeToken the code should probably check for stakeToken != rewardToken if that's the expected business behavior.

Code Location:

Listing 15: contracts/defi/StakeNft.sol (Lines 48,49)

```
32      function newPeriod(
33          uint key,
34          address stakeToken,
35          address rewardToken,
36          uint startTime,
37          uint endTime,
38          uint rewardPool
39      )
40          external
41      {
42          require(stakeToken != address(0), "STAKE_TOKEN:
                zero_address");
43
44          newStakePeriod(key, startTime, endTime, rewardPool);
45
46          // Challenge.stake is not null, means that earn is not
                null too.
47          Period storage period       = periods[msg.sender][key];
48          period.stakeToken           = stakeToken;
49          period.rewardToken          = rewardToken;
50
51          emit NewPeriod(msg.sender, key, stakeToken, rewardToken,
                startTime, endTime);
52      }
```

Listing 16: contracts/defi/StakeToken.sol (Lines 43,44)

```
29      function newPeriod(
30          uint256 key,
31          address stakeToken,
32          address rewardToken,
33          uint256 startTime,
34          uint256 endTime,
35          uint256 rewardPool
36      )
37          external
38      {
39          newStakePeriod(key, startTime, endTime, rewardPool);
40
41          // Challenge.stake is not null, means that earn is not
                null too.
42          Period storage period       = periods[msg.sender][key];
43          period.stakeToken           = stakeToken;
44          period.rewardToken          = rewardToken;
45
46          emit NewPeriod(msg.sender, key, stakeToken, rewardToken,
                startTime, endTime);
47      }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to check stakeToken != rewardToken when creating a new period.

Recommendation:

ACKNOWLEDGED: The SeaScape team acknowledged this finding.

3.16 (HAL-16) DEPRECATED METHODS - INFORMATIONAL

Description:

The `now` keyword is deprecated, use `block.timestamp` instead.

Code Location:

- ZombieFarm
  - `isActive`
  - `isStarting`
  - `startSession`

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to change all `now` occurrences to `block.timestamp`.

Recommendation:

ACKNOWLEDGED: The SeaScape team acknowledged this finding.

MANUAL TESTING

4.1 Stake

The newStake does not allow adding the same period twice:

```
>>>
>>> s.newStakePeriod(1, 111, 222, 123)
Transaction sent: 0x1bcb31376b7d04fdbbf9f65ad7382000e7514d27aa2af0c885156a904f503e47
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  Stake.newStakePeriod confirmed - Block: 4   Gas used: 125164 (1.04%)

<Transaction '0x1bcb31376b7d04fdbbf9f65ad7382000e7514d27aa2af0c885156a904f503e47'>
>>> s.newStakePeriod(1, 111, 222, 123)
Transaction sent: 0x55bda50529383b217b68ff3e3dfeed57466a2ac0ac04e5286865adafc608f11e
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  Stake.newStakePeriod confirmed (STAKE_TOKEN: period_exists) - Block: 5   Gas used: 23016 (0.19%)

<Transaction '0x55bda50529383b217b68ff3e3dfeed57466a2ac0ac04e5286865adafc608f11e'>
>>>
```

However, if startTime is zero it allows adding it again, overriding previous values.

```
>>> s.newStakePeriod(2, 0, 222, 123)
Transaction sent: 0xa9464d98131f0be39a1f3b8998063f7580f56180ff497110b02b4e239bb5fdf9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  Stake.newStakePeriod confirmed - Block: 10   Gas used: 29152 (0.24%)

<Transaction '0xa9464d98131f0be39a1f3b8998063f7580f56180ff497110b02b4e239bb5fdf9'>
>>> s.stakePeriods(a[0], 2)
(0, 222, 123, 0, 0, 0, 0, 0)
>>> s.newStakePeriod(2, 0, 333, 456)
Transaction sent: 0x34e57e1930d1b828b2efef0010728a24a17b06c0e059ead0e2f25714d8f5bcc0
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 10
  Stake.newStakePeriod confirmed - Block: 11   Gas used: 56776 (0.47%)

<Transaction '0x34e57e1930d1b828b2efef0010728a24a17b06c0e059ead0e2f25714d8f5bcc0'>
>>> s.stakePeriods(a[0], 2)
(0, 333, 456, 1, 0, 0, 0, 0)
>>>
```

Furthermore, the newStakePeriod function does allow adding a period with session ID of 0 (aka key of 0) which will always report itself as none active, a check should be added on validStakePeriodParams.

```
>>> s.newStakePeriod(0, 0, 999999999999, 123)
Transaction sent: 0x4adafce57c6bfd5c0c085f5db46efcd9a662516ca79590aa2dcd40bcca292a43
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  Stake.newStakePeriod confirmed - Block: 2   Gas used: 67588 (0.56%)

<Transaction '0x4adafce57c6bfd5c0c085f5db46efcd9a662516ca79590aa2dcd40bcca292a43'>
>>> s.newStakePeriod(1, 0, 999999999999, 123)
Transaction sent: 0xdaf99cf031963abe406cc1f27dd547742409ddd633b8c7ea84203fa4af72ce6a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  Stake.newStakePeriod confirmed - Block: 3   Gas used: 67600 (0.56%)

<Transaction '0xdaf99cf031963abe406cc1f27dd547742409ddd633b8c7ea84203fa4af72ce6a'>
>>> s.isActive(a[0], 1)
True
>>> s.isActive(a[0], 0)
False
>>>
```

Furthermore, if the reward is less than the difference between the start and end timestamps the unit (reward per second) will always be 0, in this case the reward was 1641929034 and the timestamp diff was 1641929035 - 0. This can be problematic since the period will be in seconds and probably rewards, and if rewards are not scaled.

```
>>> s.newStakePeriod(0, 0, int(time.time()), int(time.time()))
Transaction sent: 0x104bd477156db60ddb50023bde45b2379b304ad817c8cbf95262081e2a1a44bb
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  Stake.newStakePeriod confirmed - Block: 10   Gas used: 37612 (0.31%)

<Transaction '0x104bd477156db60ddb50023bde45b2379b304ad817c8cbf95262081e2a1a44bb'>
>>> s.stakePeriods(a[0], 0)
(0, 1641929030, 1641929030, 1, 0, 0, 0, 0)
>>> s.newStakePeriod(0, 0, int(time.time()), int(time.time()-1))
Transaction sent: 0xac04cbe465aa0b9a528da3763530f5acfffe0781037e53302976eb50ed12583c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 10
  Stake.newStakePeriod confirmed - Block: 11   Gas used: 26812 (0.22%)

<Transaction '0xac04cbe465aa0b9a528da3763530f5acfffe0781037e53302976eb50ed12583c'>
>>> s.stakePeriods(a[0], 0)
(0, 1641929035, 1641929034, 0, 0, 0, 0, 0)
>>>
```

The getPeriodTime function does always report endTime if the startTime is in the future. If getPeriodTime is used inside a function that does not use the whenStakePeriodActive modifier, it will be vulnerable. Function using getPeriodTime are:

- claimable: Since the staker deposit is only updated when inside the startTIme/endTime range, it will return 0 before even reaching the invalid getPeriodTime
- updatePeriodClaimable (internal): Called using the updateRewardClaimable modifier. It is used on the deposit function after the

whenStakePeriodActive. Also used on the withdraw function that will fail since no deposit was made yet. It is also called on reward function, but deposit is checked there for being not zero.

- _reward (internal): The code does check staker.deposit == 0 which will only change its value during deposit. The deposit function does call whenStakePeriodActive.

```
>>>
>>> s.getPeriodTime(now() + 10, 1234)
1234
>>>
```

The initiated function does report true even if the session is in the future and hasn't started yet:

```
>>>
>>> s.newStakePeriod(1, now() + 1000, now() + 2000 , 1234)
Transaction sent: 0xf158b5e746e8e1367efb27be2ea6d82c382021747adaff5decaf90f680080ab6
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  Stake.newStakePeriod confirmed - Block: 5   Gas used: 125203 (1.04%)

<Transaction '0xf158b5e746e8e1367efb27be2ea6d82c382021747adaff5decaf90f680080ab6'>
>>> s.initiated(a[0], 1)
True
>>>
```

Depositing into a none active period does fail:

```
>>>
>>> s.deposit(1, a[0], 1000)
Transaction sent: 0x7c198f79819508d8c025c18c36e9cd356913318f583afa51150814fdfbecd9a5
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  Stake.deposit confirmed (STAKE_TOKEN:no active period) - Block: 3   Gas used: 23240 (0.19%)

<Transaction '0x7c198f79819508d8c025c18c36e9cd356913318f583afa51150814fdfbecd9a5'>
>>>
```

4.2 VaultHandler

The `transferFromVaultToUser` does not take into consideration deflationary/inflationary tokens

4.3 StakeNFT and StakeToken

Probably it will be a good idea to check `stakeToken != rewardToken` if that's the expected business behavior

4.4 StakeToken

The code on `newPeriod` creation must check for both `stakeToken` and `rewardToken` decimals to be 18 the same as `SCALED` under `Stake` contract. Otherwise, the staking period calculations will be wrong, including the rewards.

> It does not have protection for external methods to only be called by farm

4.5 ZombieFarm

- The `now` keyword is deprecated use `block.timestamp` instead, under `isActive` , `isStarting`, `startSession`
- The `startSession` will allow creating sessions that can overlap in time.
- The `isStarting` function does not skip active sessions. It will report true for sessions that are already started but not finished. This allows `addChallengeToSession` and `addLevelRewardToSession` functions

to be called even after the session started. It will be better to rename the function to `isNotFinished` or change the code to take the `startTime` into consideration.

> The same challenge can be added into multiple sessions. However, it does not cause reward issues since reward are tracked outside. It does not cause completion issues, since the challenge itself does track the session id who is refering to.

- The `repick` feature states that a random challenge should be picked. However, the code does not perform any setting action to the `playerChallenges` mapping with a valid `sessionChallenges`
- The `addLevelRewardToSession` does allow setting the same `_data.imgId` to multiple `levelId`'s including the Grand reward. The code should check for ID duplication on the `_data` parameters.

```
>>>
>>> farm.addLevelRewardToSession(1, 1, reward, _data)
Transaction sent: 0x73b207dba7e4bca0f8d6ce8b91b0cc2bd864b5d73511eab7478bfd4074ba647e
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 178
  ZombieFarm.addLevelRewardToSession confirmed - Block: 179   Gas used: 141902 (1.18%)

<Transaction '0x73b207dba7e4bca0f8d6ce8b91b0cc2bd864b5d73511eab7478bfd4074ba647e'>
>>> farm.addLevelRewardToSession(1, 2, reward, _data)
Transaction sent: 0xb943af561f3fc398c61cc6ce530f375d7de350f6a0789f9f6b31c4f035b4592f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 179
  ZombieFarm.addLevelRewardToSession confirmed - Block: 180   Gas used: 141902 (1.18%)

<Transaction '0xb943af561f3fc398c61cc6ce530f375d7de350f6a0789f9f6b31c4f035b4592f'>
>>>
```

4.6 SingleTokenChallenge

During the constructor the code should check that both `stakeToken` and `rewardToken` decimals are equal to the `scaler` factor, otherwise the staking period calculations will be wrong including the rewards.

- The `addChallengeToSession` should check for `levelId != 0`.

CALL GRAPH

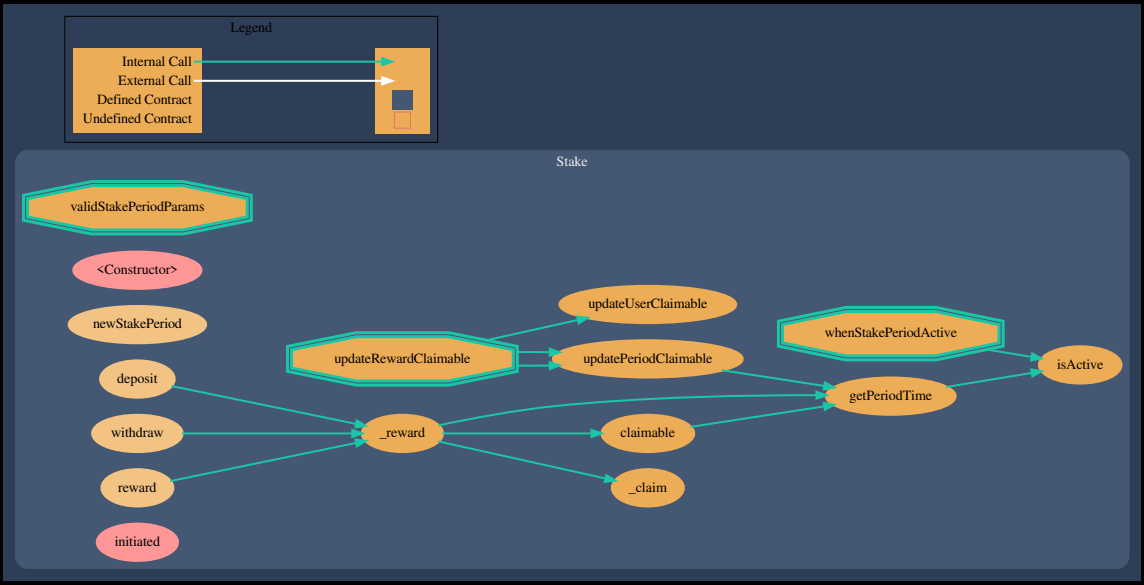

Figure 1: Stake call graph

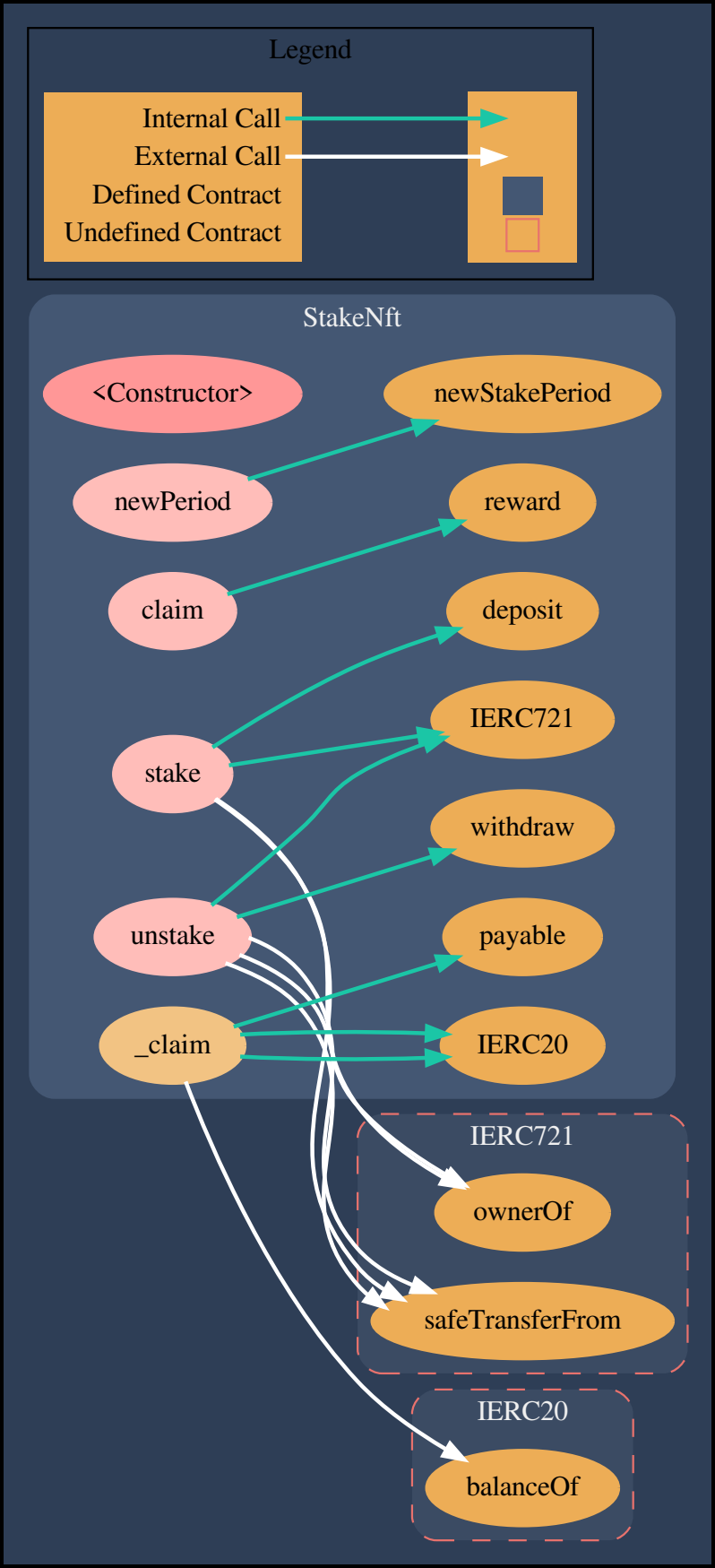

Figure 2: StakeNft call graph

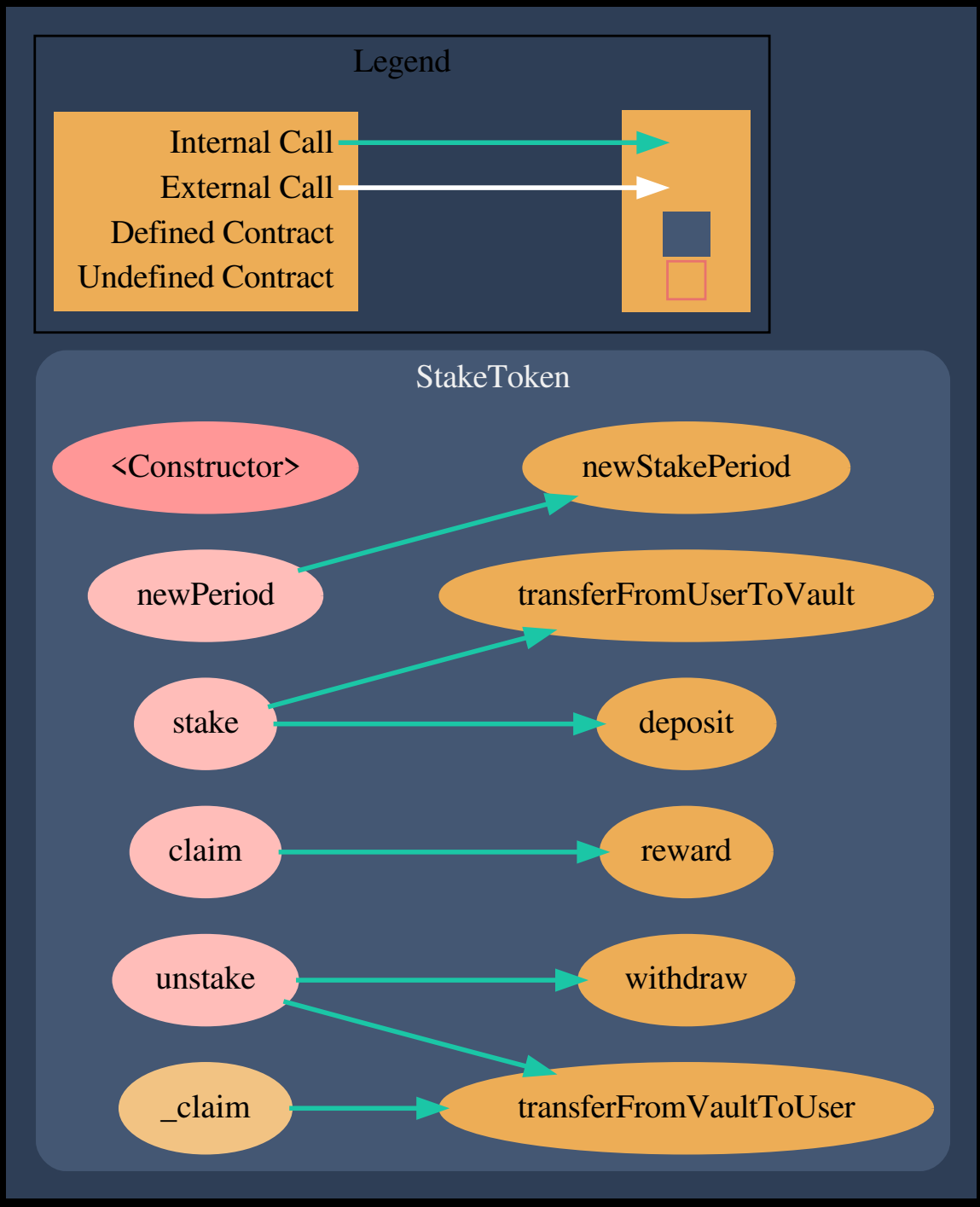

Figure 3: StakeToken call graph

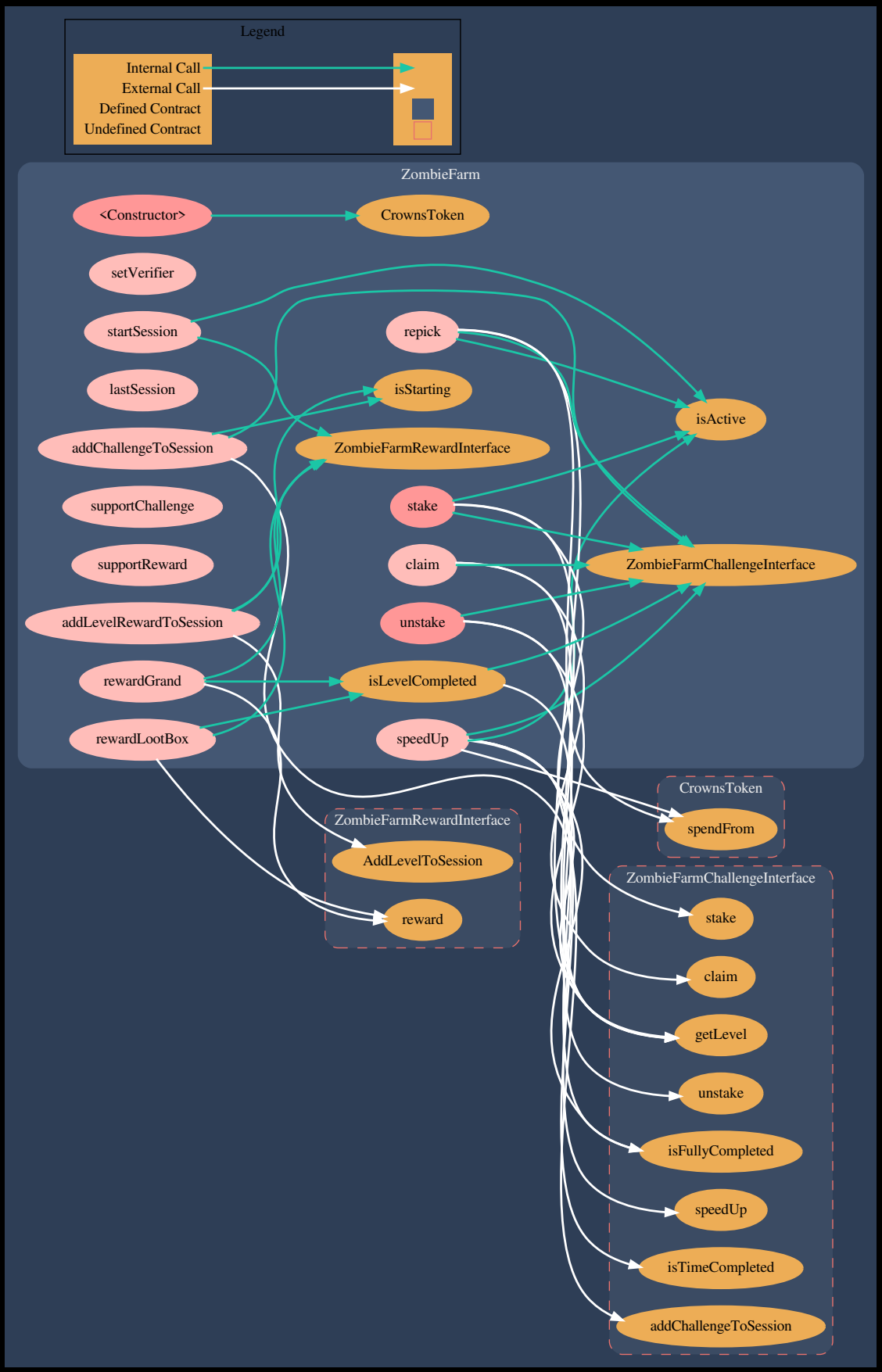

Figure 4: ZombieFarm call graph

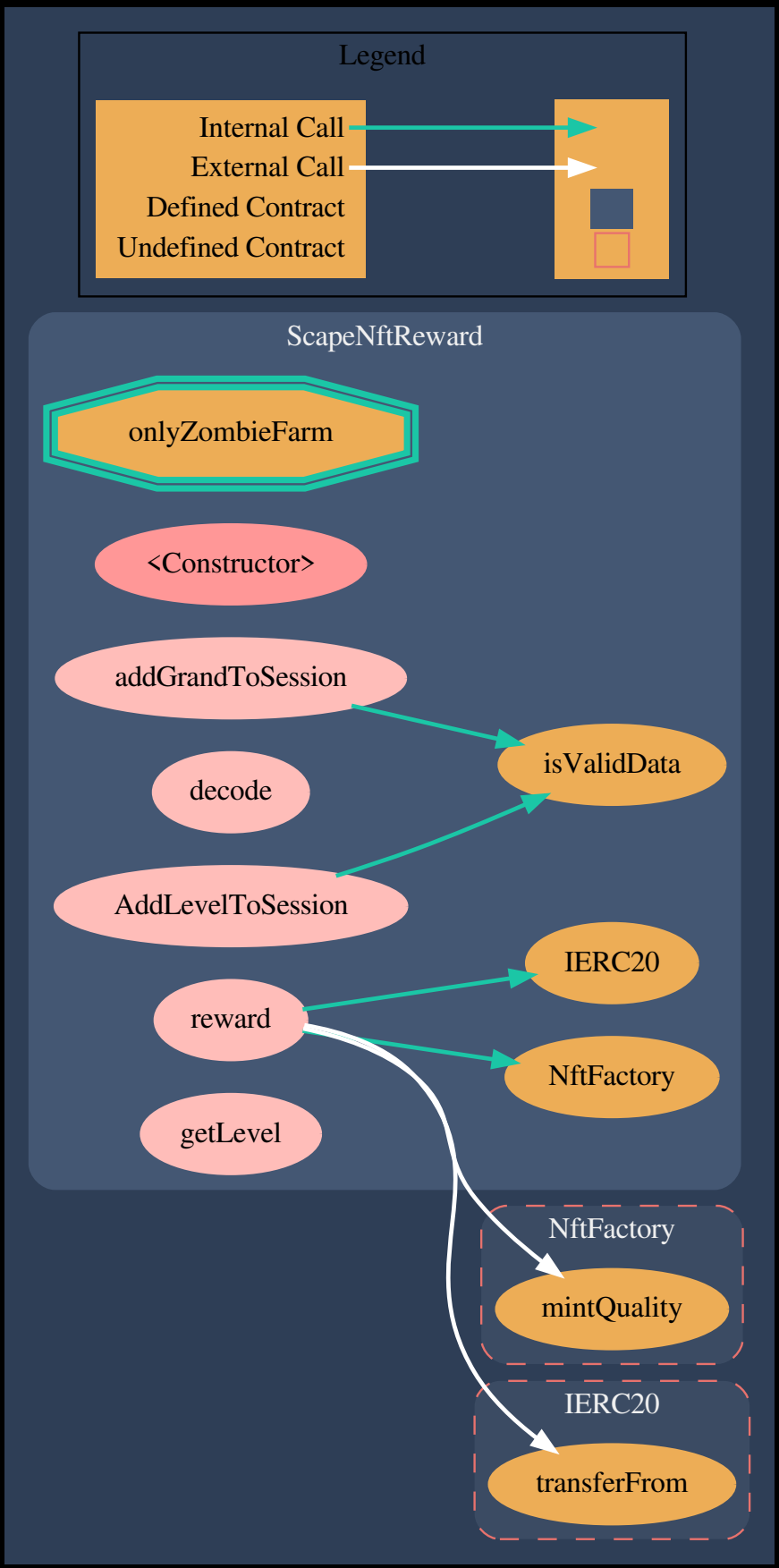

Figure 5: ScapeNftReward call graph

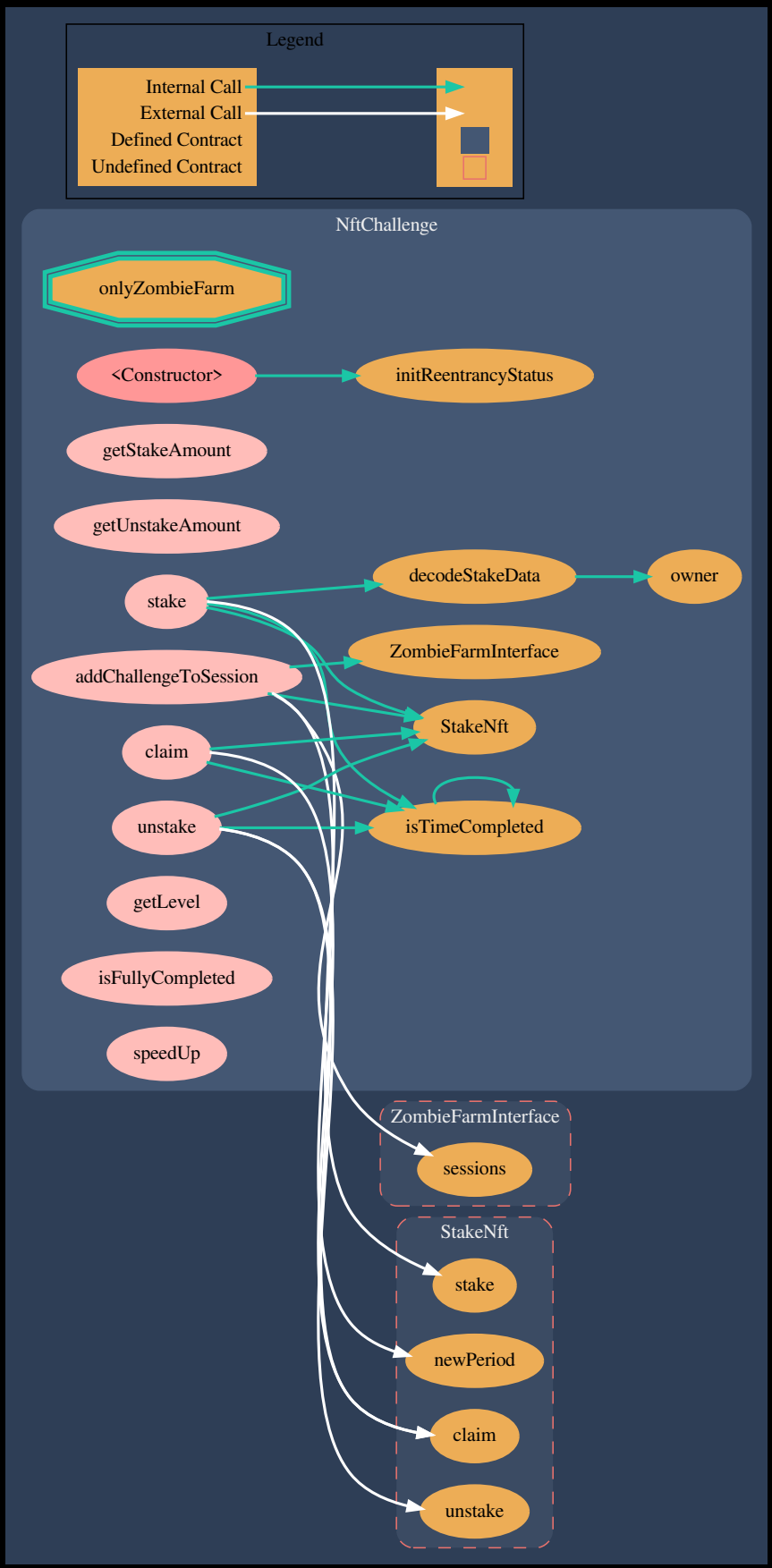

Figure 6: SingleNftChallenge call graph

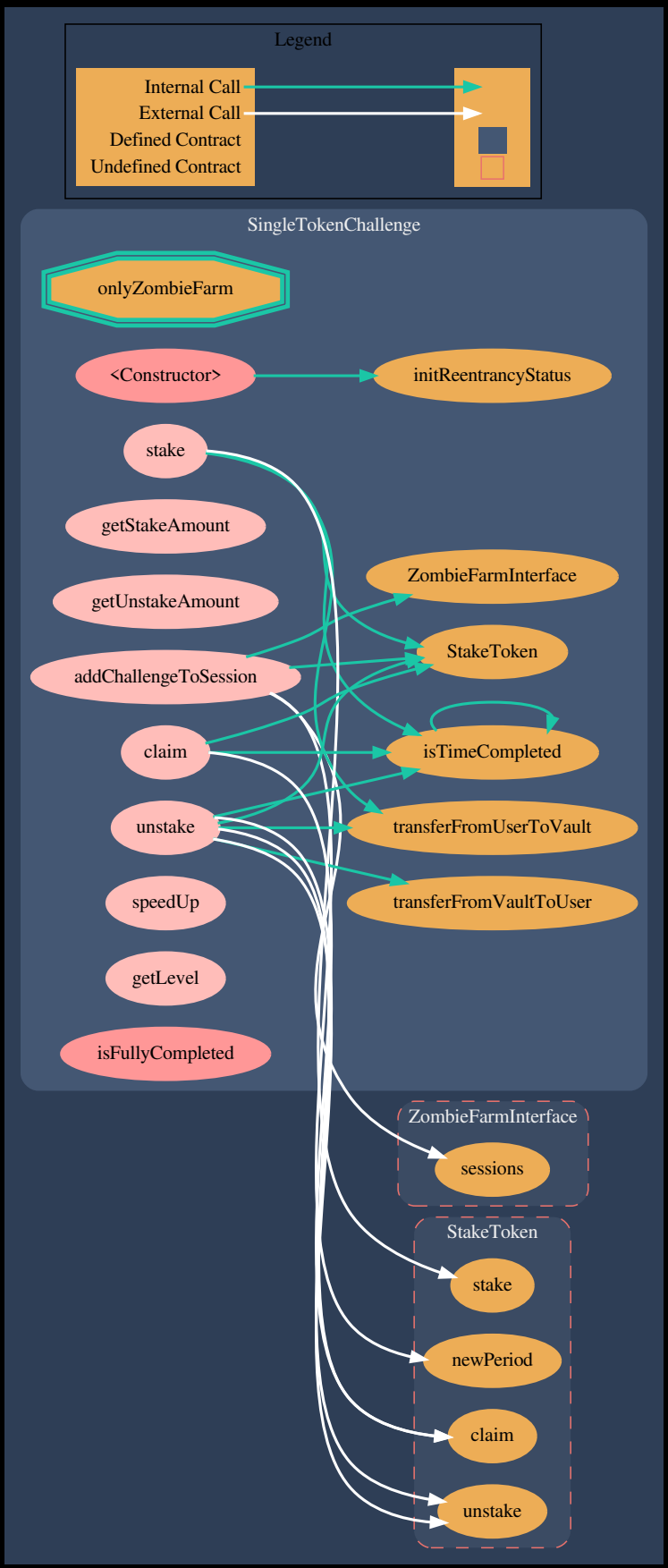

Figure 7: SingleTokenChallenge call graph

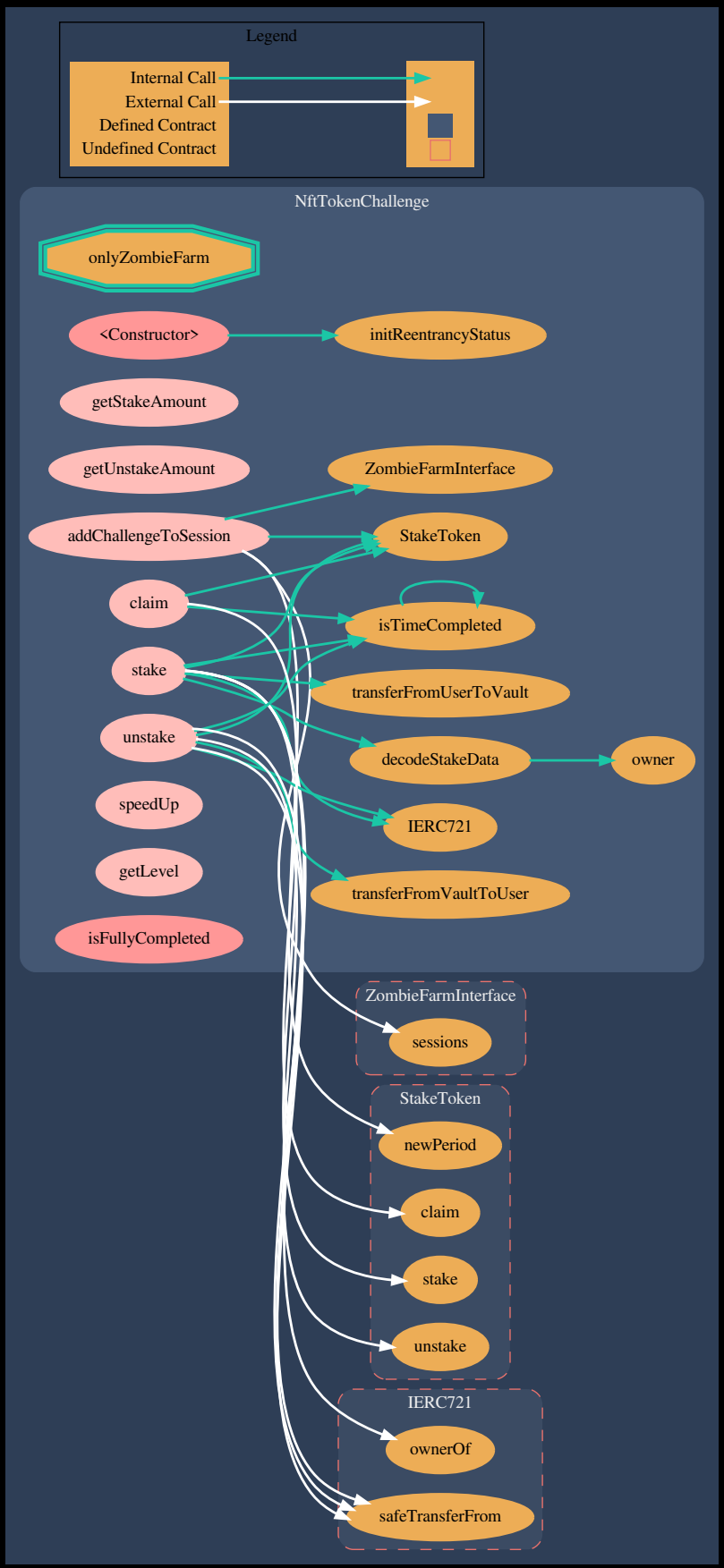

Figure 8: NftTokenChallenge call graph

AUTOMATED TESTING

6.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. 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 code-base.

Slither results:

Slither did not report any severe issue. All informative issues that were considered of importance are explained already on the report.

```
Compilation warnings/errors on contracts/game_5/ZombieFarm.sol:
Warning: Unused local variable.
   --> contracts/game_5/ZombieFarm.sol:167:9:
    |
167 |         ZombieFarmRewardInterface reward = ZombieFarmRewardInterface(grandReward);
    |         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


INFO:Detectors:
Reentrancy in ZombieFarm.addLevelRewardToSession(uint8,uint8,address,bytes) (contracts/game_5/ZombieFarm.sol#332-355):
        External calls:
        - zombieReward.AddLevelToSession(sessionId,levelId,data) (contracts/game_5/ZombieFarm.sol#350)
        State variables written after the call(s):
        - sessionRewards[sessionId][levelId] = reward (contracts/game_5/ZombieFarm.sol#352)
Reentrancy in ZombieFarm.rewardGrand(uint256) (contracts/game_5/ZombieFarm.sol#376-391):
        External calls:
        - reward.reward(sessionId,0,msg.sender) (contracts/game_5/ZombieFarm.sol#388)
        State variables written after the call(s):
        - playerRewards[sessionId][msg.sender][0] = true (contracts/game_5/ZombieFarm.sol#390)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
ZombieFarm.speedUp(uint256,uint8,address) (contracts/game_5/ZombieFarm.sol#260-281) contains a tautology or contradiction:
        - require(bool,string)(slotId >= 0 && slotId < 3,invalid slot id) (contracts/game_5/ZombieFarm.sol#261)
ZombieFarm.repick(uint256,uint8,address) (contracts/game_5/ZombieFarm.sol#283-300) contains a tautology or contradiction:
        - require(bool,string)(slotId >= 0 && slotId < 3,invalid slot id) (contracts/game_5/ZombieFarm.sol#284)
ZombieFarm.stake(uint256,uint8,address,uint8,bytes32,bytes32,bytes) (contracts/game_5/ZombieFarm.sol#404-425) contains a tautology or contradiction:
        - require(bool,string)(slotId >= 0 && slotId < 3,invalid slot id) (contracts/game_5/ZombieFarm.sol#407)
ZombieFarm.claim(uint256,uint8,address) (contracts/game_5/ZombieFarm.sol#452-466) contains a tautology or contradiction:
        - require(bool,string)(slotId >= 0 && slotId < 3,invalid slot id) (contracts/game_5/ZombieFarm.sol#453)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction
INFO:Detectors:
CrownsToken.payWaveOwing(address).payWave (contracts/crowns/erc-20/contracts/CrownsToken/CrownsToken.sol#113) shadows:
        - CrownsToken.payWave() (contracts/crowns/erc-20/contracts/CrownsToken/CrownsToken.sol#437-449) (function)
CrownsToken.allowance(address,address).owner (contracts/crowns/erc-20/contracts/CrownsToken/CrownsToken.sol#202) shadows:
        - Ownable.owner() (contracts/openzeppelin/contracts/access/Ownable.sol#35-37) (function)
CrownsToken._approve(address,address,uint256).owner (contracts/crowns/erc-20/contracts/CrownsToken/CrownsToken.sol#360) shadows:
        - Ownable.owner() (contracts/openzeppelin/contracts/access/Ownable.sol#35-37) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
ZombieFarm.constructor(address,address)._verifier (contracts/game_5/ZombieFarm.sol#129) lacks a zero-check on :
```

Figure 9: contracts/game_5/contracts_game_5_ZombieFarm_slither

```
Compilation warnings/errors on contracts/game_5/challenges/NftTokenChallenge.sol:
Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
   --> contracts/game_5/challenges/NftTokenChallenge.sol:209:54:
    |
209 |     function unstake(uint sessionId, address staker, bytes calldata data)
    |                                                      ^^^^^^^^^^^^^^^^^^


INFO:Detectors:
Reentrancy in NftTokenChallenge.unstake(uint256,address,bytes) (contracts/game_5/challenges/NftTokenChallenge.sol#209-254):
        External calls:
        - handler.unstake(sessionId,staker,sessionChallenge.stakeAmount) (contracts/game_5/challenges/NftTokenChallenge.sol#235)
        - transferFromVaultToUser(stakeToken,keepAmount,staker) (contracts/game_5/challenges/NftTokenChallenge.sol#240)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (contracts/openzeppelin/contracts/token/ERC20/SafeERC20.sol#69)
                - (success,returndata) = target.call{value: weiValue}(data) (contracts/openzeppelin/contracts/utils/Address.sol#123)
                - _token.safeTransferFrom(vault,user,amount) (contracts/game_5/helpers/VaultHandler.sol#49)
        - _nft.safeTransferFrom(address(this),staker,playerChallenge.nftId) (contracts/game_5/challenges/NftTokenChallenge.sol#245)
        - _nft.safeTransferFrom(address(this),address(0),playerChallenge.nftId) (contracts/game_5/challenges/NftTokenChallenge.sol#247)
        External calls sending eth:
        - transferFromVaultToUser(stakeToken,keepAmount,staker) (contracts/game_5/challenges/NftTokenChallenge.sol#240)
                - address(user).transfer(amount) (contracts/game_5/helpers/VaultHandler.sol#43)
                - (success,returndata) = target.call{value: weiValue}(data) (contracts/openzeppelin/contracts/utils/Address.sol#123)
        State variables written after the call(s):
        - playerChallenge.completed = true (contracts/game_5/challenges/NftTokenChallenge.sol#252)
        - playerChallenge.nftId = 0 (contracts/game_5/challenges/NftTokenChallenge.sol#253)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
Stake._reward(uint256,address) (contracts/defi/Stake.sol#224-249) uses a dangerous strict equality:
        - interest == 0 (contracts/defi/Stake.sol#233)
Stake.claimable(address,uint256,address) (contracts/defi/Stake.sol#251-275) uses a dangerous strict equality:
        - sessionCap == period.endTime && staker.rewardClaimedTime >= sessionCap (contracts/defi/Stake.sol#267)
Stake.validStakePeriodParams(uint256,uint256,uint256,uint256) (contracts/defi/Stake.sol#53-58) uses a dangerous strict equality:
        - require(bool,string)(stakePeriods[msg.sender][key].startTime == 0,STAKE_TOKEN: period_exists) (contracts/defi/Stake.sol#56)
NftTokenChallenge.stake(uint256,address,bytes) (contracts/game_5/challenges/NftTokenChallenge.sol#153-206) uses a dangerous strict equality:
        - playerChallenge.nftId == 0 (contracts/game_5/challenges/NftTokenChallenge.sol#183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in NftTokenChallenge.stake(uint256,address,bytes) (contracts/game_5/challenges/NftTokenChallenge.sol#153-206):
        External calls:
        - _nft.safeTransferFrom(staker,address(this),nftId) (contracts/game_5/challenges/NftTokenChallenge.sol#188)
```

Figure 10: contracts/game_5/challenges/contracts_game_5_challenges_-NftTokenChallenge_slither

```
INFO:Detectors:
Reentrancy in SingleTokenChallenge.stake(uint256,address,bytes) (contracts/game_5/challenges/SingleTokenChallenge.sol#137-180):
        External calls:
        - handler.stake(sessionId,staker,sessionChallenge.stakeAmount) (contracts/game_5/challenges/SingleTokenChallenge.sol#166)
        - transferFromUserToVault(stakeToken,total - sessionChallenge.stakeAmount,staker) (contracts/game_5/challenges/SingleTokenChallenge.sol#170)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (contracts/openzeppelin/contracts/token/ERC20/SafeERC20.sol#69)
                - (success,returndata) = target.call{value: weiValue}(data) (contracts/openzeppelin/contracts/utils/Address.sol#123)
                - _token.safeTransferFrom(user,vault,amount) (contracts/game_5/helpers/VaultHandler.sol#33)
        External calls sending eth:
        - transferFromUserToVault(stakeToken,total - sessionChallenge.stakeAmount,staker) (contracts/game_5/challenges/SingleTokenChallenge.sol#170)
                - (success,returndata) = target.call{value: weiValue}(data) (contracts/openzeppelin/contracts/utils/Address.sol#123)
        State variables written after the call(s):
        - playerChallenge.stakedTime = block.timestamp (contracts/game_5/challenges/SingleTokenChallenge.sol#173)
        - playerChallenge.amount = total (contracts/game_5/challenges/SingleTokenChallenge.sol#177)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
Stake._reward(uint256,address) (contracts/defi/Stake.sol#224-249) uses a dangerous strict equality:
        - interest == 0 (contracts/defi/Stake.sol#233)
Stake.claimable(address,uint256,address) (contracts/defi/Stake.sol#251-275) uses a dangerous strict equality:
        - sessionCap == period.endTime && staker.rewardClaimedTime >= sessionCap (contracts/defi/Stake.sol#267)
Stake.validStakePeriodParams(uint256,uint256,uint256,uint256) (contracts/defi/Stake.sol#53-58) uses a dangerous strict equality:
        - require(bool,string)(stakePeriods[msg.sender][key].startTime == 0,STAKE_TOKEN: period_exists) (contracts/defi/Stake.sol#56)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in SingleTokenChallenge.unstake(uint256,address,bytes) (contracts/game_5/challenges/SingleTokenChallenge.sol#182-243):
        External calls:
        - handler.claim(sessionId,staker) (contracts/game_5/challenges/SingleTokenChallenge.sol#201)
        State variables written after the call(s):
        - playerChallenge.amount = playerChallenge.amount - amount (contracts/game_5/challenges/SingleTokenChallenge.sol#210)
        - playerChallenge.stakedDuration = 0 (contracts/game_5/challenges/SingleTokenChallenge.sol#211)
        - playerChallenge.stakedTime = block.timestamp (contracts/game_5/challenges/SingleTokenChallenge.sol#212)
        - playerChallenge.completed = true (contracts/game_5/challenges/SingleTokenChallenge.sol#231)
Reentrancy in SingleTokenChallenge.unstake(uint256,address,bytes) (contracts/game_5/challenges/SingleTokenChallenge.sol#182-243):
        External calls:
        - handler.claim(sessionId,staker) (contracts/game_5/challenges/SingleTokenChallenge.sol#201)
        - handler.unstake(sessionId,staker,sessionChallenge.stakeAmount) (contracts/game_5/challenges/SingleTokenChallenge.sol#215)
        State variables written after the call(s):
        - playerChallenge.addedToPool = false (contracts/game_5/challenges/SingleTokenChallenge.sol#216)
Reentrancy in SingleTokenChallenge.unstake(uint256,address,bytes) (contracts/game_5/challenges/SingleTokenChallenge.sol#182-243):
        External calls:
```

Figure 11: contracts/game_5/challenges/contracts_game_5_challenges_-SingleTokenChallenge_slither

```
Compilation warnings/errors on contracts/game_5/challenges/SingleNftChallenge.sol:
Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/game_5/challenges/SingleNftChallenge.sol:88:29:
   |
88 |     function getStakeAmount(bytes calldata data) external override view returns (uint256) {
   |                             ^^^^^^^^^^^^^^^^^^^

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/game_5/challenges/SingleNftChallenge.sol:92:31:
   |
92 |     function getUnstakeAmount(bytes calldata data) external override view returns (uint256) {
   |                               ^^^^^^^^^^^^^^^^^^^

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
   --> contracts/game_5/challenges/SingleNftChallenge.sol:168:54:
    |
168 |     function unstake(uint sessionId, address staker, bytes calldata data)
    |                                                      ^^^^^^^^^^^^^^^^^^^


INFO:Detectors:
Stake._reward(uint256,address) (contracts/defi/Stake.sol#224-249) uses a dangerous strict equality:
        - interest == 0 (contracts/defi/Stake.sol#233)
Stake.claimable(address,uint256,address) (contracts/defi/Stake.sol#251-275) uses a dangerous strict equality:
        - sessionCap == period.endTime && staker.rewardClaimedTime >= sessionCap (contracts/defi/Stake.sol#267)
Stake.validStakePeriodParams(uint256,uint256,uint256,uint256) (contracts/defi/Stake.sol#53-58) uses a dangerous strict equality:
        - require(bool,string)(stakePeriods[msg.sender][key].startTime == 0,STAKE_TOKEN: period_exists) (contracts/defi/Stake.sol#56)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in NftChallenge.unstake(uint256,address,bytes) (contracts/game_5/challenges/SingleNftChallenge.sol#168-197):
        External calls:
        - handler.claim(sessionId,staker) (contracts/game_5/challenges/SingleNftChallenge.sol#182)
        - handler.unstake(sessionId,staker,playerChallenge.nftId,sessionChallenge.burn) (contracts/game_5/challenges/SingleNftChallenge.sol#187)
        State variables written after the call(s):
        - playerChallenge.completed = true (contracts/game_5/challenges/SingleNftChallenge.sol#190)
        - playerChallenge.nftId = 0 (contracts/game_5/challenges/SingleNftChallenge.sol#193)
        - playerChallenge.stakedTime = block.timestamp (contracts/game_5/challenges/SingleNftChallenge.sol#194)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
NftChallenge.unstake(uint256,address,bytes) (contracts/game_5/challenges/SingleNftChallenge.sol#168-197) ignores return value by handler.claim(sessionId,staker) (contracts/game_5/challenges/
SingleNftChallenge.sol#182)
```

Figure 12: contracts/game_5/challenges/contracts_game_5_challenges_-SingleNftChallenge_slither

```
Compilation warnings/errors on contracts/game_5/rewards/ScapeNftReward.sol:
Warning: Function state mutability can be restricted to pure
  --> contracts/game_5/rewards/ScapeNftReward.sol:94:5:
   |
94 |     function decode(bytes calldata data)
   |     ^ (Relevant source part starts here and spans across multiple lines).


INFO:Detectors:
ERC721._mint(address,uint256) (contracts/openzeppelin/contracts/token/ERC721/ERC721.sol#333-344) ignores return value by _holderTokens[to].add(tokenId) (contracts/openzeppelin/contracts/toke
n/ERC721/ERC721.sol#339)
ERC721._mint(address,uint256) (contracts/openzeppelin/contracts/token/ERC721/ERC721.sol#333-344) ignores return value by _tokenOwners.set(tokenId,to) (contracts/openzeppelin/contracts/token/
ERC721/ERC721.sol#341)
ERC721._burn(uint256) (contracts/openzeppelin/contracts/token/ERC721/ERC721.sol#356-374) ignores return value by _holderTokens[owner].remove(tokenId) (contracts/openzeppelin/contracts/token/
ERC721/ERC721.sol#369)
ERC721._burn(uint256) (contracts/openzeppelin/contracts/token/ERC721/ERC721.sol#356-374) ignores return value by _tokenOwners.remove(tokenId) (contracts/openzeppelin/contracts/token/ERC721/E
RC721.sol#371)
ERC721._transfer(address,address,uint256) (contracts/openzeppelin/contracts/token/ERC721/ERC721.sol#387-402) ignores return value by _holderTokens[from].remove(tokenId) (contracts/openzeppel
in/contracts/token/ERC721/ERC721.sol#396)
ERC721._transfer(address,address,uint256) (contracts/openzeppelin/contracts/token/ERC721/ERC721.sol#387-402) ignores return value by _holderTokens[to].add(tokenId) (contracts/openzeppelin/co
ntracts/token/ERC721/ERC721.sol#397)
ERC721._transfer(address,address,uint256) (contracts/openzeppelin/contracts/token/ERC721/ERC721.sol#387-402) ignores return value by _tokenOwners.set(tokenId,to) (contracts/openzeppelin/cont
racts/token/ERC721/ERC721.sol#399)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
ScapeNftReward.reward(uint256,uint8,address).owner (contracts/game_5/rewards/ScapeNftReward.sol#126) shadows:
        - Ownable.owner() (contracts/openzeppelin/contracts/access/Ownable.sol#35-37) (function)
ERC721.constructor(string,string).name (contracts/openzeppelin/contracts/token/ERC721/ERC721.sol#93) shadows:
        - ERC721.name() (contracts/openzeppelin/contracts/token/ERC721/ERC721.sol#122-124) (function)
        - IERC721Metadata.name() (contracts/openzeppelin/contracts/token/ERC721/IERC721Metadata.sol#16) (function)
ERC721.constructor(string,string).symbol (contracts/openzeppelin/contracts/token/ERC721/ERC721.sol#93) shadows:
        - ERC721.symbol() (contracts/openzeppelin/contracts/token/ERC721/ERC721.sol#129-131) (function)
        - IERC721Metadata.symbol() (contracts/openzeppelin/contracts/token/ERC721/IERC721Metadata.sol#21) (function)
SeascapeNft.setOwner(address)._owner (contracts/seascape_nft/SeascapeNft.sol#61) shadows:
        - Ownable._owner (contracts/openzeppelin/contracts/access/Ownable.sol#19) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
SeascapeNft.setFactory(address)._factory (contracts/seascape_nft/SeascapeNft.sol#65) lacks a zero-check on :
                - factory = _factory (contracts/seascape_nft/SeascapeNft.sol#66)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in SeascapeNft.mint(address,uint256,uint8) (contracts/seascape_nft/SeascapeNft.sol#48-59):
        External calls:
        - _safeMint(_to,_tokenId) (contracts/seascape_nft/SeascapeNft.sol#51)
                - (success,returndata) = target.call{value: weiValue}(data) (contracts/openzeppelin/contracts/utils/Address.sol#123)
                - returndata = to.functionCall(abi.encodeWithSelector(IERC721Receiver(to).onERC721Received.selector,_msgSender(),from,tokenId,_data),ERC721: transfer to non ERC721Receiver im
plementer) (contracts/openzeppelin/contracts/token/ERC721/ERC721.sol#441-447)
        External calls sending eth:
```

Figure 13: contracts/game_5/rewards/contracts_game_5_rewards_-ScapeNftReward_slither

```
INFO:Detectors:
Stake._reward(uint256,address) (contracts/defi/Stake.sol#224-249) uses a dangerous strict equality:
        - interest == 0 (contracts/defi/Stake.sol#233)
Stake.claimable(address,uint256,address) (contracts/defi/Stake.sol#251-275) uses a dangerous strict equality:
        - sessionCap == period.endTime && staker.rewardClaimedTime >= sessionCap (contracts/defi/Stake.sol#267)
Stake.validStakePeriodParams(uint256,uint256,uint256,uint256) (contracts/defi/Stake.sol#53-58) uses a dangerous strict equality:
        - require(bool,string)(stakePeriods[msg.sender][key].startTime == 0,STAKE_TOKEN: period_exists) (contracts/defi/Stake.sol#56)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Stake.withdraw(uint256,address,uint256) (contracts/defi/Stake.sol#136-157) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(amount > 0 && staker.deposit >= amount,STAKE_TOKEN: stake amount zero) (contracts/defi/Stake.sol#142)
Stake._reward(uint256,address) (contracts/defi/Stake.sol#224-249) uses timestamp for comparisons
        Dangerous comparisons:
        - interest == 0 (contracts/defi/Stake.sol#233)
Stake.claimable(address,uint256,address) (contracts/defi/Stake.sol#251-275) uses timestamp for comparisons
        Dangerous comparisons:
        - sessionCap == period.endTime && staker.rewardClaimedTime >= sessionCap (contracts/defi/Stake.sol#267)
Stake.isActive(uint256,uint256) (contracts/defi/Stake.sol#292-298) uses timestamp for comparisons
        Dangerous comparisons:
        - (block.timestamp >= startTime && block.timestamp <= endTime) (contracts/defi/Stake.sol#297)
Stake.isActive(address,uint256) (contracts/defi/Stake.sol#303-308) uses timestamp for comparisons
        Dangerous comparisons:
        - (block.timestamp >= period.startTime && block.timestamp <= period.endTime) (contracts/defi/Stake.sol#307)
Stake.initiated(address,uint256) (contracts/defi/Stake.sol#313-318) uses timestamp for comparisons
        Dangerous comparisons:
        - (block.timestamp <= period.endTime) (contracts/defi/Stake.sol#317)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (contracts/openzeppelin/contracts/utils/Address.sol#26-35) uses assembly
        - INLINE ASM (contracts/openzeppelin/contracts/utils/Address.sol#33)
Address._functionCallWithValue(address,bytes,uint256,string) (contracts/openzeppelin/contracts/utils/Address.sol#119-140) uses assembly
        - INLINE ASM (contracts/openzeppelin/contracts/utils/Address.sol#132-135)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['0.6.7', '^0.6.0', '^0.6.2', '^0.6.7']
        - 0.6.7 (contracts/defi/Stake.sol#1)
        - 0.6.7 (contracts/defi/StakeToken.sol#1)
        - 0.6.7 (contracts/game_5/helpers/VaultHandler.sol#1)
```

Figure 14: contracts/defi/contracts_defi_StakeToken_slither

```
INFO:Detectors:
Stake._reward(uint256,address) (contracts/defi/Stake.sol#224-249) uses a dangerous strict equality:
        - interest == 0 (contracts/defi/Stake.sol#233)
Stake.claimable(address,uint256,address) (contracts/defi/Stake.sol#251-275) uses a dangerous strict equality:
        - sessionCap == period.endTime && staker.rewardClaimedTime >= sessionCap (contracts/defi/Stake.sol#267)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in StakeNft.stake(uint256,address,uint256,uint256) (contracts/defi/StakeNft.sol#55-73):
        External calls:
        - nft.safeTransferFrom(stakerAddr,address(this),id) (contracts/defi/StakeNft.sol#70)
        - deposit(key,stakerAddr,amount) (contracts/defi/StakeNft.sol#72)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (contracts/openzeppelin/contracts/token/ERC20/SafeERC20.sol#69)
                - (success,returndata) = target.call{value: weiValue}(data) (contracts/openzeppelin/contracts/utils/Address.sol#123)
                - IERC20(_token).safeTransferFrom(vault,stakerAddr,interest) (contracts/defi/StakeNft.sol#115)
        External calls sending eth:
        - deposit(key,stakerAddr,amount) (contracts/defi/StakeNft.sol#72)
                - address(stakerAddr).transfer(interest) (contracts/defi/StakeNft.sol#109)
                - (success,returndata) = target.call{value: weiValue}(data) (contracts/openzeppelin/contracts/utils/Address.sol#123)
        Event emitted after the call(s):
        - Deposit(msg.sender,stakerAddr,key,amount) (contracts/defi/Stake.sol#133)
                - deposit(key,stakerAddr,amount) (contracts/defi/StakeNft.sol#72)
        - Reward(msg.sender,stakerAddr,key,interest) (contracts/defi/Stake.sol#246)
                - deposit(key,stakerAddr,amount) (contracts/defi/StakeNft.sol#72)
Reentrancy in StakeNft.unstake(uint256,address,uint256,bool) (contracts/defi/StakeNft.sol#75-95):
        External calls:
        - nft.safeTransferFrom(address(this),stakerAddr,id) (contracts/defi/StakeNft.sol#89)
        - nft.safeTransferFrom(address(this),address(0),id) (contracts/defi/StakeNft.sol#91)
        - withdraw(key,stakerAddr,weights[msg.sender][key][id]) (contracts/defi/StakeNft.sol#94)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (contracts/openzeppelin/contracts/token/ERC20/SafeERC20.sol#69)
                - (success,returndata) = target.call{value: weiValue}(data) (contracts/openzeppelin/contracts/utils/Address.sol#123)
                - IERC20(_token).safeTransferFrom(vault,stakerAddr,interest) (contracts/defi/StakeNft.sol#115)
        External calls sending eth:
        - withdraw(key,stakerAddr,weights[msg.sender][key][id]) (contracts/defi/StakeNft.sol#94)
                - address(stakerAddr).transfer(interest) (contracts/defi/StakeNft.sol#109)
                - (success,returndata) = target.call{value: weiValue}(data) (contracts/openzeppelin/contracts/utils/Address.sol#123)
        Event emitted after the call(s):
        - Reward(msg.sender,stakerAddr,key,interest) (contracts/defi/Stake.sol#246)
                - withdraw(key,stakerAddr,weights[msg.sender][key][id]) (contracts/defi/StakeNft.sol#94)
        - Withdraw(msg.sender,stakerAddr,key,amount) (contracts/defi/Stake.sol#156)
                - withdraw(key,stakerAddr,weights[msg.sender][key][id]) (contracts/defi/StakeNft.sol#94)
```

Figure 15: contracts/defi/contracts_defi_StakeNft_slither

```
INFO:Detectors:
Stake._reward(uint256,address) (contracts/defi/Stake.sol#224-249) uses a dangerous strict equality:
        - interest == 0 (contracts/defi/Stake.sol#233)
Stake.claimable(address,uint256,address) (contracts/defi/Stake.sol#251-275) uses a dangerous strict equality:
        - sessionCap == period.endTime && staker.rewardClaimedTime >= sessionCap (contracts/defi/Stake.sol#267)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Stake.withdraw(uint256,address,uint256) (contracts/defi/Stake.sol#136-157) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(amount > 0 && staker.deposit >= amount,STAKE_TOKEN: stake amount zero) (contracts/defi/Stake.sol#142)
Stake._reward(uint256,address) (contracts/defi/Stake.sol#224-249) uses timestamp for comparisons
        Dangerous comparisons:
        - interest == 0 (contracts/defi/Stake.sol#233)
Stake.claimable(address,uint256,address) (contracts/defi/Stake.sol#251-275) uses timestamp for comparisons
        Dangerous comparisons:
        - sessionCap == period.endTime && staker.rewardClaimedTime >= sessionCap (contracts/defi/Stake.sol#267)
Stake.isActive(uint256,uint256) (contracts/defi/Stake.sol#292-298) uses timestamp for comparisons
        Dangerous comparisons:
        - (block.timestamp >= startTime && block.timestamp <= endTime) (contracts/defi/Stake.sol#297)
Stake.isActive(address,uint256) (contracts/defi/Stake.sol#303-308) uses timestamp for comparisons
        Dangerous comparisons:
        - (block.timestamp >= period.startTime && block.timestamp <= period.endTime) (contracts/defi/Stake.sol#307)
Stake.initiated(address,uint256) (contracts/defi/Stake.sol#313-318) uses timestamp for comparisons
        Dangerous comparisons:
        - (block.timestamp <= period.endTime) (contracts/defi/Stake.sol#317)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Stake._claim(uint256,address,uint256) (contracts/defi/Stake.sol#222) is never used and should be removed
Stake._reward(uint256,address) (contracts/defi/Stake.sol#224-249) is never used and should be removed
Stake.deposit(uint256,address,uint256) (contracts/defi/Stake.sol#109-134) is never used and should be removed
Stake.newStakePeriod(uint256,uint256,uint256,uint256) (contracts/defi/Stake.sol#80-100) is never used and should be removed
Stake.reward(uint256,address) (contracts/defi/Stake.sol#159-174) is never used and should be removed
Stake.updatePeriodClaimable(uint256) (contracts/defi/Stake.sol#185-209) is never used and should be removed
Stake.updateUserClaimable(uint256,Stake.StakeUser) (contracts/defi/Stake.sol#212-220) is never used and should be removed
Stake.withdraw(uint256,address,uint256) (contracts/defi/Stake.sol#136-157) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.6.7 (contracts/defi/Stake.sol#1) allows old versions
solc-0.6.7 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

Figure 16: contracts/defi/contracts_defi_Stake_slither

THANK YOU FOR CHOOSING

// HALBORN