PlayGround Labs - Kapital-DAO

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 28th, 2022 - April 14th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 6
CONTACTS 6
1 EXECUTIVE OVERVIEW 7
1.1 INTRODUCTION 8
1.2 AUDIT SUMMARY 8
1.3 TEST APPROACH & METHODOLOGY 8
RISK METHODOLOGY 9
1.4 SCOPE 11
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 12
3 FINDINGS & TECH DETAILS 13
3.1 (HAL-01) PROPOSAL LACKS MULTIPLE IMPORTANT LOGIC CHECKS - MEDIUM 15
Description 15
Code Location 15
Risk Level 16
Recommendation 16
Remediation Plan 17
3.2 (HAL-02) UNCHECKED TRANSFER - LOW 20
Description 20
Code Location 20
Risk Level 24
Recommendation 24
Remediation Plan 25
3.3 (HAL-03) MISSING RE-ENTRANCY PROTECTION - LOW 26
Description 26

Code Location 26

Risk Level 27

Recommendation 28

Remediation Plan 28

3.4 (HAL-04) UNINITIALIZED PROPOSE COOLDOWN - LOW 29

Description 29

Code Location 29

Risk Level 30

Recommendation 30

Remediation Plan 30

3.5 (HAL-05) EXTERNAL FUNCTION CALLS WITHIN LOOP - LOW 31

Description 31

Code Location 31

Risk Level 32

Recommendation 32

Reference 32

Remediation Plan 32

3.6 (HAL-06) IGNORE RETURN VALUES - LOW 33

Description 33

Code Location 33

Risk Level 34

Recommendation 34

Remediation Plan 34

3.7 (HAL-07) WEAK GOVERNANCE OWNERSHIP TRANSFER - LOW 35

Description 35

PoC Steps 35

Risk Level 35

Recommendation 35

Remediation Plan 36

3.8 (HAL-08) MISSING LEGITIMACY OF VOTE CASTER - LOW 37

Description 37

Code Location 37

Risk Level 38

Recommendation 38

Remediation Plan 39

3.9 (HAL-09) USAGE OF BLOCK-TIMESTAMP - LOW 40

Description 40

Code Location 40

Risk Level 41

Recommendation 41

Reference 41

Remediation Plan 42

3.10 (HAL-10) MISSING ZERO-ADDRESS CHECK - INFORMATIONAL 43

Description 43

Code Location 43

Risk Level 44

Recommendation 44

Remediation Plan 44

3.11 (HAL-11) DIVIDE BEFORE MULTIPLY - INFORMATIONAL 45

Description 45

Code Location 45

Risk Level 45

Recommendation 45

Remediation Plan 45

3.12 (HAL-12) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL 46

Description 46

Code Location 46

Risk Level 46

Recommendation 46

Remediation Plan 47

3.13 (HAL-13) EXPONENTIATION IS MORE COSTLY - INFORMATIONAL 48

Description 48

Example 48

Code Location 48

Risk Level 48

Recommendation 48

Remediation Plan 49

3.14 (HAL-14) USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS - INFORMATIONAL 50

Description 50

Code Location 50

Risk Level 51

Proof of Concept 51

Risk Level 52

Recommendation 52

Remediation Plan 53

3.15 (HAL-15) CACHE ARRAY LENGTH IN FOR LOOPS CAN SAVE GAS - INFORMATIONAL 54

Description 54

Code Location 54

Risk Level 55

Recommendation 55

Remediation Plan 56

4 AUTOMATED TESTING 57

4.1 STATIC ANALYSIS REPORT 58

Description 58

Results 58

4.2 AUTOMATED SECURITY SCAN 60

Description 60

Results 60

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/06/2022	Juned Ansari
0.2	Document Edits	04/08/2022	Juned Ansari
0.3	Document Edits	04/11/2022	Juned Ansari
0.4	Draft Review	04/12/2022	Gabi Urrutia
1.0	Remediation Plan	05/17/2022	Juned Ansari
1.1	Remediation Plan Review	05/19/2022	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
Juned Ansari	Halborn	Juned.Ansari@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

PlayGround Labs engaged Halborn to conduct a security audit on their smart contracts beginning on March 28th, 2022 and ending on April 14th, 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 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 the functions of the Kapital-DAO contract work as intended.
- Identify potential security issues within the smart contracts.

In summary, Halborn identified some security risks that were mostly addressed by the PlayGround Labs 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 Kapital-DAO contract solidity 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.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX).
- Static Analysis of security for scoped contract, and imported functions. (Slither).
- Testnet deployment (Remix IDE).

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. 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.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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

1.4 SCOPE

IN-SCOPE : Kapital-DAO-Halborn-Audit

IN-SCOPE COMMIT : 53d86b8933c63105112818e15705ea0d77954c47

OUT-OF-SCOPE : External libraries, test-helpers and economics attacks.

FIXED-COMMIT : 35fb92524b83ff8197a7127f7c9819317ac7ea92

EXECUTIVE OVERVIEW

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	8	6

LIKELIHOOD

IMPACT

🟨	🟨	🟥	🟥	🟪
🟩 (HAL-03)	🟨	🟨	🟥	🟥
🟩 (HAL-02) (HAL-07) (HAL-09)	🟩 (HAL-05)	🟨	🟨 (HAL-01)	🟥
⬜ (HAL-10) (HAL-11)	🟩 (HAL-08)	🟩 (HAL-04) (HAL-06)	🟨	🟨
⬜ (HAL-12) (HAL-13) (HAL-14) (HAL-15)	⬜	🟩	🟩	🟨

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
PROPOSAL LACKS MULTIPLE IMPORTANT LOGIC CHECKS	Medium	SOLVED - 05/17/2022
UNCHECKED TRANSFER	Low	SOLVED - 05/17/2022
MISSING RE-ENTRANCY PROTECTION	Low	SOLVED - 05/17/2022
UNINITIALIZED PROPOSE COOLDOWN	Low	SOLVED - 05/17/2022
EXTERNAL FUNCTION CALLS WITHIN LOOP	Low	SOLVED - 05/17/2022
IGNORE RETURN VALUES	Low	RISK ACCEPTED
WEAK GOVERNANCE OWNERSHIP TRANSFER	Low	SOLVED - 05/17/2022
MISSING LEGITIMACY OF VOTE CASTER	Low	SOLVED - 05/17/2022
USAGE OF BLOCK-TIMESTAMP	Low	RISK ACCEPTED
MISSING ZERO-ADDRESS CHECK	Informational	SOLVED - 05/17/2022
DIVIDE BEFORE MULTIPLY	Informational	SOLVED - 05/17/2022
POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED - 05/17/2022
EXPONENTIATION IS MORE COSTLY	Informational	SOLVED - 05/17/2022
USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS	Informational	SOLVED - 05/17/2022
CACHE ARRAY LENGTH IN FOR LOOPS CAN SAVE GAS	Informational	SOLVED - 05/17/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) PROPOSAL LACKS MULTIPLE IMPORTANT LOGIC CHECKS - MEDIUM

Description:

In Governance.sol contracts, the propose function lacks the multiple logical checks listed below.
- Missing to check if the length of the provided targets is equal to values, data length or not. Thus allowing the incorrect submission of the proposals.
- No maximum length of targets is defined, which leads to too many actions in a proposal.
- No logic is implemented in the proposer if the proposer already has an active or pending proposal in the system before adding a new proposal.
- If a description argument is missing from the proposal, it should be used to log the description of the proposal.
- Insufficient event emitting, only emitting msg.sender, latestProposalID , and timestamp instead, all essential arguments supplied to the function.

Code Location:

Listing 1: Governance.sol

```
    function propose(
        address[] memory targets,
        uint256[] memory values,
        bytes[] memory data,
        bool[] memory isDelegateCall,
        WeightSources memory weightSources
    ) external returns (uint256) {
        // Ensure msg.sender has enough voting weight
        (uint256 weightKAP, uint256 weightLP) = getWeights(
 ↳ weightSources);
        require(
            weightKAP + convertLP(weightLP) >= threshold,
            "Governance: Insufficient weight"
        );
        // Make sure haven't already created a proposal within cooldown period, Propose CoolDown
        uint256 timestamp = block.timestamp;
        require(
            timestamp >= latestPropose[msg.sender] + proposeCooldown,
            "Governance: Propose Cooldown"
        );

        // Add new proposal
        latestProposalID++;
        Proposal storage proposal = proposals[latestProposalID];
        proposal.transactParamsHash = getTransactParamsHash(
            targets,
            values,
            data,
            isDelegateCall
        );
        proposal.proposeTime = SafeCast.toUint64(timestamp);
        proposal.priceCumulativeLast = _cumulative();

        // Update msg.sender's latestProposal
        latestPropose[msg.sender] = timestamp;

        emit ProposalCreated(msg.sender, latestProposalID, timestamp);
        return latestProposalID;
    }
```

Risk Level:

Likelihood - 4
Impact - 3

Recommendation:

Consider adding require checks to ensure array lengths are validated, add a description argument to the above function, and emit the same to log the proposal description for readability; it also emits the events for the remaining arguments, similar to the code shared below. In addition, it is recommended to implement logic to check the proposal status (for example, the status of the proposal could be Pending, Active, Canceled, Defeated, Succeeded, Queued, Expired, Executed, etc.) to the msg.sender, before adding a new proposal.

Listing 2

```
    function propose(
        address[] memory targets,
        uint256[] memory values,
        bytes[] memory data,
        bool[] memory isDelegateCall,
        WeightSources memory weightSources,
        string memory description
    ) external returns (uint256) {
...
require(targets.length > 0 && targets.length == values.length &&
↳ targets.length == data.length && targets.length == isDelegateCall.
↳ length, "Governor: proposal function information arity mismatch");
require(targets.length <= max_operation, "Governor: too many
↳ actions");
...
emit ProposalCreated(msg.sender, latestProposalID, timestamp,
↳ targets, values, data, isDelegateCall, weightSources, description)
↳ ;
...
```

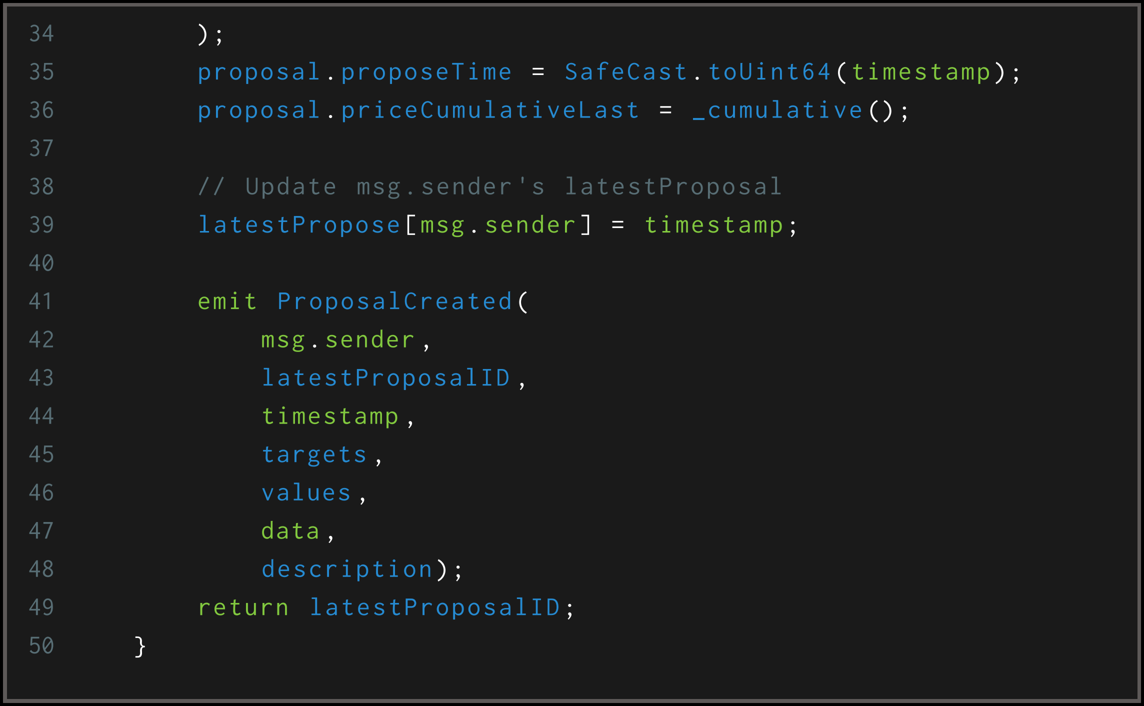
Remediation Plan:

SOLVED: The Playground labs team solved the above issue in commit 35fb92524b83ff8197a7127f7c9819317ac7ea92. As a result, the team added additional logical checks to the propose function, set proposeCooldown to 3 days (instead of 0 before) to protect against proposal spam, and added a proposal description field and relevant information to the event emitted.

Furthermore, the team claims that the use of proposeCooldown is sufficient because fast origin-address change is prohibited by the staking requirement and the new delegation-change procedure. Also, the team does not want to limit the target length, as different function calls will use significantly different amounts of gas. Finally, the team claims that the front-end UI will include features such as estimated gas usage and estimated transaction success or failure.

Listing 3: Updated propose

```
    function propose(
        address[] memory targets,
        uint256[] memory values,
        bytes[] memory data,
        string memory description,
        WeightSources memory weightSources
    ) external returns (uint256) {
        // Ensure msg.sender has enough voting weight
        (uint256 weightKAP, uint256 weightLP) = getWeights(weightSources);
        require(
            weightKAP + convertLP(weightLP) >= threshold,
            "Governance: Insufficient weight"
        );
        // Make sure haven't already created a proposal within cooldown period, Propose CoolDown
        uint256 timestamp = block.timestamp;
        require(
            timestamp >= latestPropose[msg.sender] + proposeCooldown,
            "Governance: Propose Cooldown"
        );
        // Logic check on proposal data
        uint256 targetsLength = targets.length;
        require(targetsLength > 0, "Governance: Invalid data");
        require(targetsLength == values.length, "Governance: Invalid data");
        require(targetsLength == data.length, "Governance: Invalid data");
        require(!(bytes(description).length == 0), "Governance: No description");

        // Add new proposal
        latestProposalID++;
        Proposal storage proposal = proposals[latestProposalID];
        proposal.transactParamsHash = getTransactParamsHash(
            targets,
            values,
            data
```



```
        );
        proposal.proposeTime = SafeCast.toUint64(timestamp);
        proposal.priceCumulativeLast = _cumulative();

        // Update msg.sender's latestProposal
        latestPropose[msg.sender] = timestamp;

        emit ProposalCreated(
            msg.sender,
            latestProposalID,
            timestamp,
            targets,
            values,
            data,
            description);
        return latestProposalID;
    }
```

3.2 (HAL-02) UNCHECKED TRANSFER - LOW

Description:

In RewardsLocker.sol, Staking.sol, Vesting.sol contracts, return values from external transfer calls are not checked. It should be noted that the token is not reverted on failure and returns false. If one of these tokens is used, a deposit would not be reverted if the transfer failed and an attacker could deposit tokens for free.

Code Location:

Listing 4: RewardsLocker.sol (Line 107)

```
91      function collectRewards(uint256 lockAgreementId) external {
92          LockAgreement storage lockAgreement = lockAgreements[msg.
 ↳ sender][
93              lockAgreementId
94          ];
95          // make sure the beneficiary waits before collecting the
 ↳ rewards
96          require(
97              block.timestamp >= lockAgreement.availableTimestamp,
98              "Collection too early"
99          );
100         // make sure the beneficiary has not already collected the
 ↳  rewards
101         require(!lockAgreement.collected, "Already collected");
102         // set `collected` to true, so the beneficiary cannot
 ↳ withdraw again
103         lockAgreement.collected = true;
104         // update voting weight
105         weightKAP[msg.sender] -= lockAgreement.amount;
106         // transfer `amount` KAP to the beneficiary
107         kapToken.transfer(msg.sender, lockAgreement.amount);
108     }
```

Listing 5: RewardsLocker.sol (Line 120)

```
    function transferKap(address to, uint256 amount) external {
        bool senderIsGovernance = (msg.sender ==
            governanceRegistry.governance());
        bool authorized = senderIsGovernance || hasRole(KAP_SAVER, msg.sender);
        require(authorized, "Access denied");
        kapToken.transfer(to, amount);
    }
```

Listing 6: Staking.sol (Line 456)

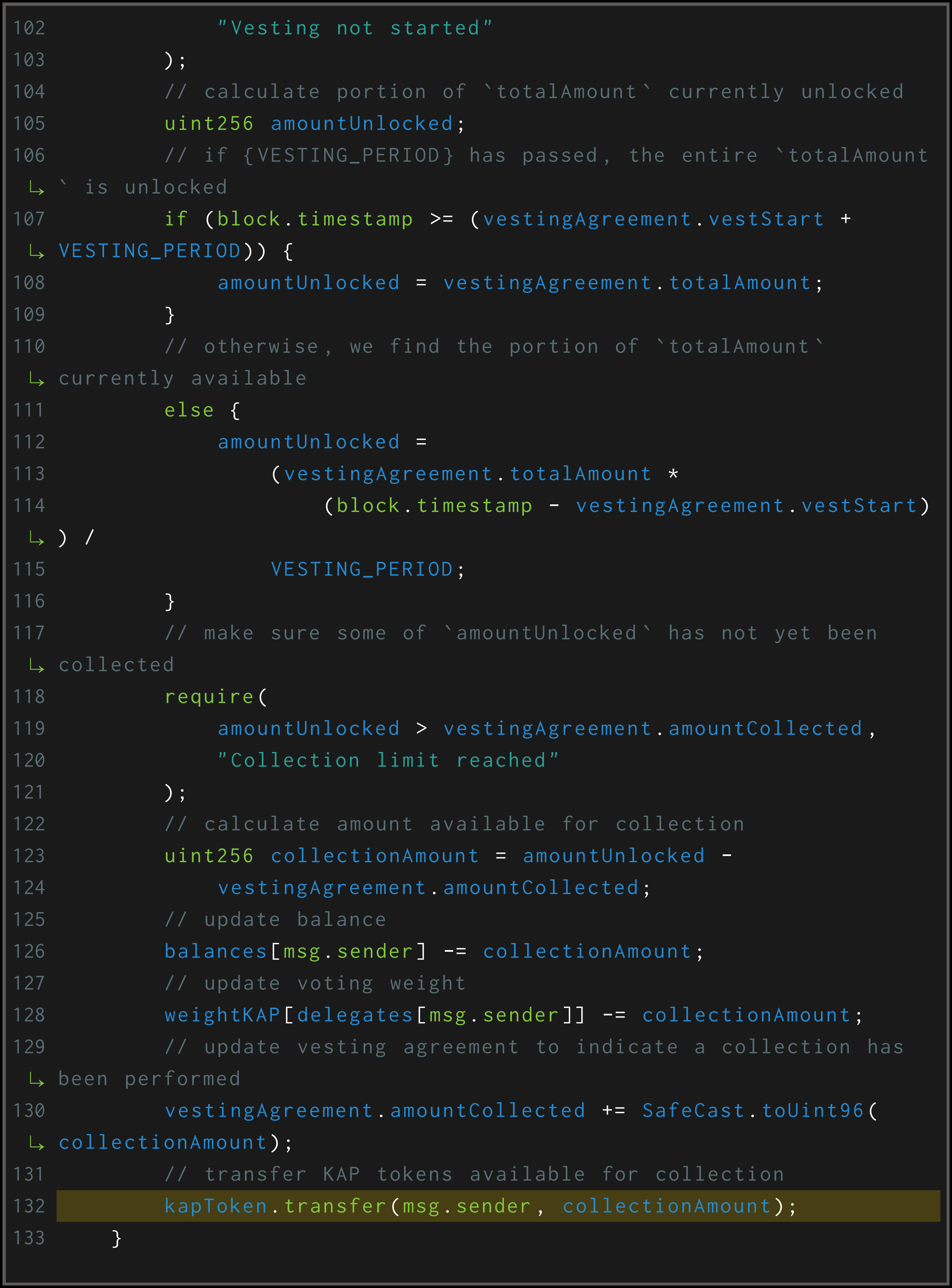
```
    function _transferFromAndReturnAddAmount(
        address staker,
        uint256 inputAmount
    ) internal returns (uint256) {
        uint256 previousBalance = asset.balanceOf(address(this));
        asset.transferFrom(staker, address(this), inputAmount);
        return asset.balanceOf(address(this)) - previousBalance;
    }
```

Listing 7: Staking.sol (Line 248)

```
    function unstake(uint256 removeAmount, uint256 stakingAgreementId)
        external
    {
        // update {multipliedTotalRewardsPerWeightLastSync} if necessary
        if (block.timestamp > lastSync) {
            sync();
        }

        Staker storage staker = stakers[msg.sender];
        StakingAgreement storage stakingAgreement = staker.stakingAgreements[
            stakingAgreementId
        ];

        require(
            (removeAmount > 0) && (removeAmount <= stakingAgreement.amount),
            "Staking: Invalid amount"
        );
        require(
            block.timestamp >= stakingAgreement.lockEnd,
            "Staking: Too early"
        );

        asset.transfer(msg.sender, removeAmount);
        staker.totalAmount -= Math.toUint112(removeAmount);
        stakingAgreement.amount -= Math.toUint112(removeAmount);

        uint256 unstakeWeight = _calculateStakeWeight(
            stakingAgreement.lockEnd - stakingAgreement.lockStart,
            removeAmount
        );
        totalStakingWeight -= unstakeWeight;
        staker.totalWeight -= Math.toUint136(unstakeWeight);
        // looking at {claimRewards}, the above line has instantaneously
        // decreased `claimedRewards` by
        // `(unstakeWeight * multipliedTotalRewardsPerWeightLastSync) / REWARDS_PER_WEIGHT_MULTIPLIER`.
        // we therefore need to give this amount back to the user in the form
        // of adding to `staker.addRewards`.
        staker.addRewards +=
            (unstakeWeight * multipliedTotalRewardsPerWeightLastSync) /
            REWARDS_PER_WEIGHT_MULTIPLIER;

        emit Unstake(msg.sender, removeAmount);
    }
```

Listing 8: Vesting.sol (Line 132)

```
    function collect(uint256 vestingAgreementId) external {
        VestingAgreement storage vestingAgreement = vestingAgreements[
            msg.sender
        ][vestingAgreementId];
        // make sure the vesting period has started
        require(
            block.timestamp > vestingAgreement.vestStart,
```



```
            "Vesting not started"
        );
        // calculate portion of `totalAmount` currently unlocked
        uint256 amountUnlocked;
        // if {VESTING_PERIOD} has passed, the entire `totalAmount` is unlocked
        if (block.timestamp >= (vestingAgreement.vestStart + VESTING_PERIOD)) {
            amountUnlocked = vestingAgreement.totalAmount;
        }
        // otherwise, we find the portion of `totalAmount` currently available
        else {
            amountUnlocked =
                (vestingAgreement.totalAmount *
                    (block.timestamp - vestingAgreement.vestStart)) /
                VESTING_PERIOD;
        }
        // make sure some of `amountUnlocked` has not yet been collected
        require(
            amountUnlocked > vestingAgreement.amountCollected,
            "Collection limit reached"
        );
        // calculate amount available for collection
        uint256 collectionAmount = amountUnlocked -
            vestingAgreement.amountCollected;
        // update balance
        balances[msg.sender] -= collectionAmount;
        // update voting weight
        weightKAP[delegates[msg.sender]] -= collectionAmount;
        // update vesting agreement to indicate a collection has been performed
        vestingAgreement.amountCollected += SafeCast.toUint96(collectionAmount);
        // transfer KAP tokens available for collection
        kapToken.transfer(msg.sender, collectionAmount);
    }
```

Listing 9: Vesting.sol (Line 75)

```
69     function createVestingAgreement(
70         address beneficiary,
71         uint256 vestStart,
72         uint256 amount
73     ) external onlyRole(VESTING_CREATOR) {
74         // caller provides KAP for the vesting agreement
75         kapToken.transferFrom(msg.sender, address(this), amount);
76         // update balance
77         balances[beneficiary] += amount;
78         // update delegate voting weight
79         weightKAP[delegates[beneficiary]] += amount;
80         // push a new vesting agreement for the beneficiary
81         vestingAgreements[beneficiary].push(
82             VestingAgreement({
83                 vestStart: SafeCast.toUint64(vestStart),
84                 totalAmount: SafeCast.toUint96(amount),
85                 amountCollected: SafeCast.toUint96(0)
86             })
87         );
88     }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to use SafeERC20 or make sure the return values of transfer and transferFrom are checked. For example, the success check below can ensure a revert on failure.

Listing 10

```
        bool success = asset.transferFrom(staker, address(this), inputAmount);
        if (!success) {
            revert TransferFailed();
        }
```

Remediation Plan:

SOLVED: The Playground labs team solved the above issue in commit 35fb92524b83ff8197a7127f7c9819317ac7ea92. As a result, the team now uses OpenZeppelin's SafeERC20 to perform the transfers. However, the team claims that the tokens used in the Kapital DAO (KAP token and Uniswap V2 Pair) revert on failure, and otherwise return a hardcoded value of true in the above code.

3.3 (HAL-03) MISSING RE-ENTRANCY PROTECTION - LOW

Description:

One of the contracts included in the scope of Playground Labs Kapital-DAO was identified as missing a nonReentrant guard. In this function, persistent state read/write after an external call is identified, making it vulnerable to a Reentrancy attack.

- The Staking.sol contract function unstake is missing nonReentrant guard.

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

Code Location:

Listing 11: Staking.sol (Lines 248,249,256,257,263-265)

```
226     function unstake(uint256 removeAmount, uint256 stakingAgreementId)
227         external
228     {
229         // update {multipliedTotalRewardsPerWeightLastSync} if necessary
230         if (block.timestamp > lastSync) {
231             sync();
232         }
233
234         Staker storage staker = stakers[msg.sender];
235         StakingAgreement storage stakingAgreement = staker.stakingAgreements[
236             stakingAgreementId
```

```
        ];

        require(
            (removeAmount > 0) && (removeAmount <= stakingAgreement.amount),
            "Staking: Invalid amount"
        );
        require(
            block.timestamp >= stakingAgreement.lockEnd,
            "Staking: Too early"
        );

        asset.transfer(msg.sender, removeAmount);
        staker.totalAmount -= Math.toUint112(removeAmount);
        stakingAgreement.amount -= Math.toUint112(removeAmount);

        uint256 unstakeWeight = _calculateStakeWeight(
            stakingAgreement.lockEnd - stakingAgreement.lockStart,
            removeAmount
        );
        totalStakingWeight -= unstakeWeight;
        staker.totalWeight -= Math.toUint136(unstakeWeight);
        // looking at {claimRewards}, the above line has instantaneously
        // decreased `claimedRewards` by
        // `(unstakeWeight * multipliedTotalRewardsPerWeightLastSync) / REWARDS_PER_WEIGHT_MULTIPLIER`.
        // we therefore need to give this amount back to the user in the form
        // of adding to `staker.addRewards`.
        staker.addRewards +=
            (unstakeWeight * multipliedTotalRewardsPerWeightLastSync) /
            REWARDS_PER_WEIGHT_MULTIPLIER;

        emit Unstake(msg.sender, removeAmount);
    }
```

Risk Level:

Likelihood - 1

Impact - 4

Recommendation:

Change the code to follow the checks-effects-interactions pattern and use ReentrancyGuard via the nonReentrant modifier.

Remediation Plan:

SOLVED: The Playground labs team solved the above issue in commit 35fb92524b83ff8197a7127f7c9819317ac7ea92. As a result, the code now follows the checks-effects-interactions pattern. However, the team claims that the tokens used in Kapital DAO (KAP token and Uniswap V2 Pair) are not vulnerable to reentrancy into above code.

3.4 (HAL-04) UNINITIALIZED PROPOSE COOLDOWN - LOW

Description:

In the Governance.sol contract, the proposeCooldown state variable is not initialized, it defaults to 0 value, and the variable is considered in the other calculation progresses, i.e., require(timestamp >= latestPropose [msg.sender] + proposeCooldown, "Governance: Propose Cooldown"); in the propose function. If a variable must be initialized to zero, explicitly set it to zero to improve code readability.

Code Location:

Listing 12: Governance.sol (Line 23)

```
23  uint24 public proposeCooldown;
```

Listing 13: Governance.sol (Line 154)

```
138     function propose(
139         address[] memory targets,
140         uint256[] memory values,
141         bytes[] memory data,
142         bool[] memory isDelegateCall,
143         WeightSources memory weightSources
144     ) external returns (uint256) {
145         // Ensure msg.sender has enough voting weight
146         (uint256 weightKAP, uint256 weightLP) = getWeights(
  ↳ weightSources);
147         require(
148             weightKAP + convertLP(weightLP) >= threshold,
149             "Governance: Insufficient weight"
150         );
151         // Make sure haven't already created a proposal within
  ↳ cooldown period, Propose CoolDown
152         uint256 timestamp = block.timestamp;
153         require(
            timestamp >= latestPropose[msg.sender] + proposeCooldown,
            "Governance: Propose Cooldown"
        );

        // Add new proposal
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

If is recommended to initialize all variables in the same function, either in the constructor or in a custom init method. However, using uninitialized variables and expecting them to have a value could cause unexpected behaviors in the flow of execution.

Remediation Plan:

SOLVED: The Playground labs team solved the above issue in the commit 35fb92524b83ff8197a7127f7c9819317ac7ea92. As a result, the team initializes proposeCooldown to 3 days (same as the voting period) so that the voting process is never overwhelmed by repeated proposals.

Listing 14: Constructor Initialize propseCooldown

```
        // implicitly set propseCooldown to be the voting period
        proposeCooldown = _waitTo.endVote - _waitTo.startVote;
```

3.5 (HAL-05) EXTERNAL FUNCTION CALLS WITHIN LOOP - LOW

Description:

External calls within a loop increase Gas usage or can lead to a denial of service attack. In the Governance.sol contract functions discovered there is a for loop on the i variable that iterates through the weightSources.kapSources.length and weightSources.lpSources.length array length, and this loop has external calls within a loop. If this integer evaluates to extremely large numbers, this can cause a DoS.

Code Location:

Listing 15: Governance.sol (Lines 104,96-98)

```
88     function getWeights(WeightSources memory weightSources)
89         public
90         view
91         returns (uint256 weightKAP, uint256 weightLP)
92     {
93         // Calc KAP voting weight
94         for (uint256 i = 0; i < weightSources.kapSources.length; i
 ↳ ++) {
95             if (weightSources.kapSources[i]) {
96                 weightKAP += IKAPSource(weightSourcesKAP[i]).
 ↳ weightKAP(
97                     msg.sender
98                 );
99             }
100        }
101        // Calc LP voting weight
102        for (uint256 i = 0; i < weightSources.lpSources.length; i
 ↳ ++) {
103            if (weightSources.lpSources[i]) {
104                weightLP += ILPSource(weightSourcesLP[i]).weightLP
 ↳ (msg.sender);
105            }
106        }
107    }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It is recommended that you set the maximum length over which a for loop can iterate. If possible, use the pull over push strategy for external calls.

Reference:

External Calls Recommendation

Remediation Plan:

SOLVED: The Playground labs team solved the above issue in commit 35fb92524b83ff8197a7127f7c9819317ac7ea92. The team modifies the WeightSources and getWeights structs; as a result, the user can now choose specific array indices in weightSourcesKAP and weightSourcesLP . Furthermore, the team added that in the unlikely case that weightSourcesKAP and weightSourcesLP are very long, the user can choose a relatively small number of WeightSources to loop through.

3.6 (HAL-06) IGNORE RETURN VALUES - LOW

Description:

The return value of an external call is not stored in a local or state variable. In the Transactor.sol contract, there is an instance where an external method is called, and the return value is ignored.

Code Location:

Listing 16: Transactor.sol (Lines 39,41)

```
23     function _transact(
24         address[] memory targets,
25         uint256[] memory values,
26         bytes[] memory data,
27         bool[] memory isDelegateCall
28     ) internal {
29         require(targets.length > 0, "Invalid array length");
30         require(targets.length == values.length, "Array length
↳ mismatch");
31         require(targets.length == data.length, "Array length
↳ mismatch");
32         require(
33             targets.length == isDelegateCall.length,
34             "Array length mismatch"
35         );
36 
37         for (uint256 i = 0; i < targets.length; i++) {
38             if (isDelegateCall[i]) {
39                 Address.functionDelegateCall(targets[i], data[i]);
40             } else {
41                 Address.functionCallWithValue(targets[i], data[i],
↳  values[i]);
42             }
43         }
44     }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Add return value checking to prevent an unexpected contract crash. Checking the return value will help to handle exceptions in a better way.

Remediation Plan:

RISK ACCEPTED: The Playground labs team accept the risk of this finding. Furthermore, the team claims that the team does not have any function-specific return value due to not having information about which functions can be called in advance. However, the team added that the Address contract confirms that success == true and reverts otherwise.

3.7 (HAL-07) WEAK GOVERNANCE OWNERSHIP TRANSFER - LOW

Description:

The supplied newGovernance is not being validated before the transfer of ownership, even though the governance access control is in place. If configured incorrectly, it will lock all governance functionality.

PoC Steps:

Listing 17: GovernanceRegistry.sol (Lines 29,30)

```
28     function changeGovernance(address newGovernance) external {
29         require(msg.sender == governance, "Only governance");
30         governance = newGovernance;
31     }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Consider validating that the new governance address is different from address zero. Furthermore, two-step approvals must be set to avoid setting the wrong addresses. The first function will store an address in a global variable, and the second function will confirm the new address if msg.sender equals the new address, proving that the new owner has access to the correct private key.

Remediation Plan:

SOLVED: The Playground labs team solved the above issue in commit 35fb92524b83ff8197a7127f7c9819317ac7ea92. As a result, the team added a two-step governance change process. Furthermore, the team also added additional validation of the new governance address.

Listing 18: Updated changeGovernance

```
    function changeGovernance(address newGovernance) external {
        require(msg.sender == governance, "Only governance");
        require(
            newGovernance != address(0) && newGovernance != governance,
            "Invalid governance"
        );

        IGovernance _newGovernance = IGovernance(newGovernance);
        require(_newGovernance.votingPeriod() > 0);

        appointedNewGovernance = newGovernance;
    }
```

3.8 (HAL-08) MISSING LEGITIMACY OF VOTE CASTER - LOW

Description:

In the Governance.sol contract, it is noted that the vote function lacks the legitimacy of msg.sender if it is a valid voter. As a result, an unknown EOA or contract may cast a vote; therefore, the result of the vote can be manipulated, although it is unlikely since there are no benefits for said voter.

Code Location:

Listing 19: Governance.sol

```
    function vote(
        uint256 proposalID,
        bool yay,
        WeightSources memory weightSources
    ) external {
        Proposal storage proposal = proposals[proposalID];

        // Enforce voting window, Voting Window
        require(_checkVoteWindow(proposal), "Governance: Voting window");

        // Mark msg.sender as having voted, Already Voted
        require(!proposal.hasVoted[msg.sender], "Governance: Already voted");
        proposal.hasVoted[msg.sender] = true;

        (uint256 weightKAP, uint256 weightLP) = getWeights(weightSources);

        // Add to vote counts
        require(weightLP <= type(uint112).max, "Governance: uint112(weightLP)");
        if (yay) {
            proposal.yaysKAP += SafeCast.toUint96(weightKAP);
            proposal.yaysLP += uint112(weightLP);
        } else {
            proposal.naysKAP += SafeCast.toUint96(weightKAP);
            proposal.naysLP += uint112(weightLP);
        }

        // Record that `msg.sender` last voted at this timestamp
        lastVoted[msg.sender] = block.timestamp;

        emit Voted(msg.sender, proposalID, yay, weightKAP, weightLP);
    }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to implement a valid whitelist of members who can cast a vote on a proposal. Otherwise, validate msg.sender and ensure that only valid EOA interacts with purpose. Consider adding the validSender modifier below to avoid the above issue.

Listing 20

```
    function isContract() public view returns(bool){
      uint32 size;
      address a = msg.sender;
      assembly {
        size := extcodesize(a)
      }
      require(size==0);
    }


    modifier validSender(address sender) {
        require(!sender.isContract() && (tx.origin == msg.sender), "Only-EOA");
        _;
    }
```

Remediation Plan:

SOLVED: The Playground labs team solved the above issue in commit 35fb92524b83ff8197a7127f7c9819317ac7ea92. As a solution, the team a descriptive requirement require(weightKAP > 0 || weightLP > 0, " Governance: Zero weight") to prevent zero-weight accounts from voting unnecessarily. Furthermore, the team claims that the team allows contracts, particularly multisig wallets, to stake and vote. And the only requirement to propose is to meet the threshold that the team has currently initialized at 0.65% of the total KAP supply, and there is no whitelist of proposals.

3.9 (HAL-09) USAGE OF BLOCK-TIMESTAMP - LOW

Description:

During a manual review, the use of block.timestamp in some Playground Labs Kapital-DAO contracts were observed. Contract developers should note that this does not mean the current time. Miners can influence the value of block.timestamp to some degree, so testers should be warned that this may come at some risk if miners collude in time manipulation to influence price oracles. It is important to follow the 15-second rule, i.e., if the contract is not based on an interval of less than 15-seconds, it is fine to use block.timestamp.

Code Location:

Listing 21: Vesting.sol

```
#101: block.timestamp > vestingAgreement.vestStart,
#107: if (block.timestamp >= (vestingAgreement.vestStart + VESTING_PERIOD)) {
#114: (block.timestamp - vestingAgreement.vestStart)) /
#148: block.timestamp > oldDelegateLastVoted + votingPeriod,
```

Listing 22: Governance.sol

```
#152: uint256 timestamp = block.timestamp;
#187: uint256 timeElapsed = block.timestamp - proposal.proposeTime;
#224: lastVoted[msg.sender] = block.timestamp;
#239: uint256 timeElapsed = block.timestamp - proposal.proposeTime;
#323: emit ProposalExecuted(msg.sender, proposalID, block.timestamp);
#350: (block.timestamp - proposal.proposeTime);
```

Listing 23: Staking.sol

```
#148: (block.timestamp <= (lastStaked[voter] + votingPeriod))
#161: if (block.timestamp <= rewardsStart) {
#182: if (block.timestamp > lastSync) {
#185: lastStaked[msg.sender] = block.timestamp;
#230: if (block.timestamp > lastSync) {
#244: block.timestamp >= stakingAgreement.lockEnd,
#288: if (block.timestamp > lastSync) {
#295: if (block.timestamp <= rewardsStart) {
#320: require(block.timestamp > lastSync, "Staking: Already syncd");
#323: if (block.timestamp > rewardsRules[rewardsRuleIndex].timeEnd) {
#419: if (block.timestamp <= rewardsRule.timeEnd) {
#470: uint256 lockStart = block.timestamp <= rewardsStart
```

Listing 24: RewardsLocker.sol

```
#97: block.timestamp >= lockAgreement.availableTimestamp,
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to follow the 15-second rule, i.e., if the time-dependent event can vary by 15 seconds and maintain integrity, it is safe to use a block.timestamp.

Reference:

Ethereum Yellow Paper

Remediation Plan:

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

3.10 (HAL-10) MISSING ZERO-ADDRESS CHECK - INFORMATIONAL

Description:

Several instances found where the address validation is missing. A zero address validation failure was found when assigning user-supplied address values to state variables directly.

- In contract GovernanceRegistry.sol:
  - changeGovernance lacks a zero address check on newGovernance.
  - constructor lacks a zero address check on initialGovernance.
- In contract MultisigFund.sol:
  - constructor lacks a zero address check on _multisig.
- In contract RewardsLocker.sol:
  - constructor lacks a zero address check on _kapStakingPool, _kapEthStakingPool.
- In contract Vesting.sol:
  - constructor lacks a zero address check on _teamMultisig.

Code Location:

Zero Address Validation is missing before assigning addresses to these state variables.

Listing 25: GovernanceRegistry.sol

```
governance = initialGovernance (#20)
governance = newGovernance (#30)
```

Listing 26: MultisigFund.sol

```
multisig = _multisig (#18)
```

Listing 27: RewardsLocker.sol

```
kapStakingPool = _kapStakingPool (#34)
kapEthStakingPool = _kapEthStakingPool (#35)
```

Listing 28: Vesting.sol

```
teamMultisig = _teamMultisig (#32)
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Although administrative restrictions are imposed on this function due to role-based access controls (RBAC), it is recommended that you add proper address validation when assigning user-supplied input to a variable. This could be as simple as using the following statement:

Listing 29

```
require(address_input != 0, "Address is zero")
```

Remediation Plan:

SOLVED: The Playground labs team solved the above issue in commit 35fb92524b83ff8197a7127f7c9819317ac7ea92. As a result, the team added zero address checks for the above code.

3.11 (HAL-11) DIVIDE BEFORE MULTIPLY - INFORMATIONAL

Description:

Solidity's integer division could be truncated. As a result, precision loss can sometimes be avoided by multiplying before dividing, although the manual implementation of the precision/decimal calculation is taken care of by the developer. In the set of smart contracts, there is an instance where the division is done before the multiplication.

Code Location:

Listing 30: Governance.sol (Lines 349,350)

```
349         uint256 priceETH = (_cumulative() - proposal.priceCumulativeLast) /
350             (block.timestamp - proposal.proposeTime);
351         uint256 reserveKAP = Math.sqrt(k * priceETH);
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Consider performing multiplication before division to ensure precision in results when using non-floating-point data types.

Remediation Plan:

SOLVED: The Playground labs team solved the above issue in commit 35fb92524b83ff8197a7127f7c9819317ac7ea92. As a result, the contract now performs multiplication before division.

3.12 (HAL-12) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In public functions, array arguments are immediately copied into memory, while external functions can read directly from calldata. Reading calldata is cheaper than allocating memory. Public functions need to write arguments to memory because public functions can be called internally. Internal calls are passed internally via pointers to memory. Thus, the function expects its arguments to be located in memory when the compiler generates the code for an internal function.
Also, methods do not necessarily have to be public if they are only called within the contract; in such case, they should be marked as internal.

Code Location:

Below are the smart contracts and their corresponding functions affected:
Staking.sol:
getStakingAgreementsLength()

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider as much as possible to declare external variables instead of public variables. As for best practices, you should use external if you expect the function to only be called externally and use public if you need to call the function internally. In short, public functions can be accessed by everyone, external functions can only be accessed externally, and internal functions can only be called within the contract.

Remediation Plan:

SOLVED: The Playground labs team solved the above issue in commit 35fb92524b83ff8197a7127f7c9819317ac7ea92. As a result, the team has changed getStakingAgreementsLength() to external.

3.13 (HAL-13) EXPONENTIATION IS MORE COSTLY - INFORMATIONAL

Description:

Exponentiation is more expensive than the following implementation.

Example:

Listing 31

```
1e18 is more cheap than 10**18.
```

Code Location:

Listing 32: Token.sol (Line 16)

```
15     constructor() ERC20("Kapital DAO Token", "KAP") {
16         uint256 uiKapTotalSupply = 10**9;
17         _mint(msg.sender, uiKapTotalSupply * (10**decimals()));
18     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider replacing the 10**a with 1ea.

Remediation Plan:

SOLVED: The Playground labs team solved the above issue in commit 35fb92524b83ff8197a7127f7c9819317ac7ea92. As a result, the team replaced 10**9 with 1e9.

3.14 (HAL-14) USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS - INFORMATIONAL

Description:

In the loop below, the variable i is incremented using i++. It is known that, in loops, using ++i costs less gas per iteration than i++.

Code Location:

Listing 33: Governance.sol (Lines 94,102)

```
88      function getWeights(WeightSources memory weightSources)
89          public
90          view
91          returns (uint256 weightKAP, uint256 weightLP)
92      {
93          // Calc KAP voting weight
94          for (uint256 i = 0; i < weightSources.kapSources.length; i
  ↳ ++) {
95              if (weightSources.kapSources[i]) {
96                  weightKAP += IKAPSource(weightSourcesKAP[i]).
  ↳ weightKAP(
97                      msg.sender
98                  );
99              }
100         }
101         // Calc LP voting weight
102         for (uint256 i = 0; i < weightSources.lpSources.length; i
  ↳ ++) {
103             if (weightSources.lpSources[i]) {
104                 weightLP += ILPSource(weightSourcesLP[i]).weightLP
  ↳ (msg.sender);
105             }
106         }
107     }
```

Listing 34: Transactor.sol (Line 37)

```
23     function _transact(
24         address[] memory targets,
25         uint256[] memory values,
26         bytes[] memory data,
27         bool[] memory isDelegateCall
28     ) internal {
29         require(targets.length > 0, "Invalid array length");
30         require(targets.length == values.length, "Array length mismatch");
31         require(targets.length == data.length, "Array length mismatch");
32         require(
33             targets.length == isDelegateCall.length,
34             "Array length mismatch"
35         );
36 
37         for (uint256 i = 0; i < targets.length; i++) {
38             if (isDelegateCall[i]) {
39                 Address.functionDelegateCall(targets[i], data[i]);
40             } else {
41                 Address.functionCallWithValue(targets[i], data[i], values[i]);
42             }
43         }
44     }
```

Risk Level:

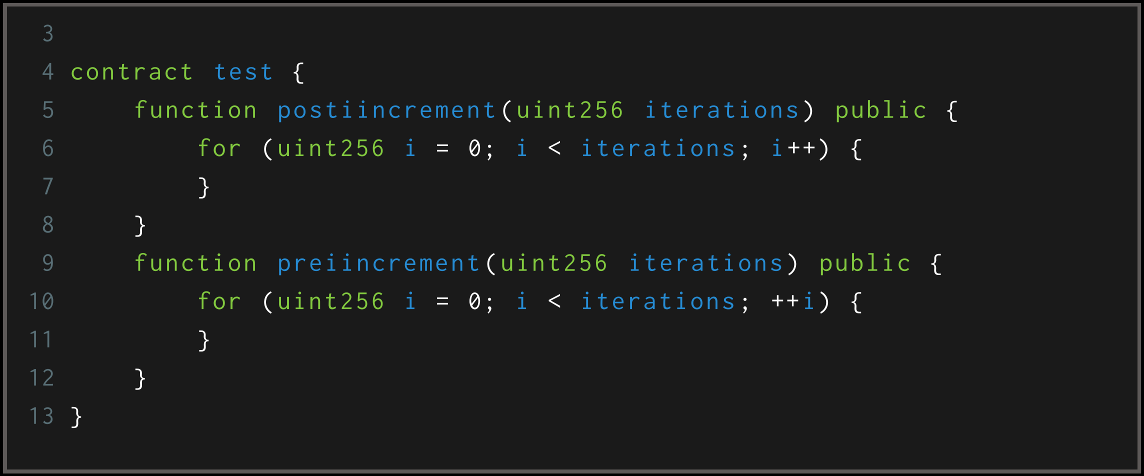
Likelihood - 1
Impact - 1

Proof of Concept:

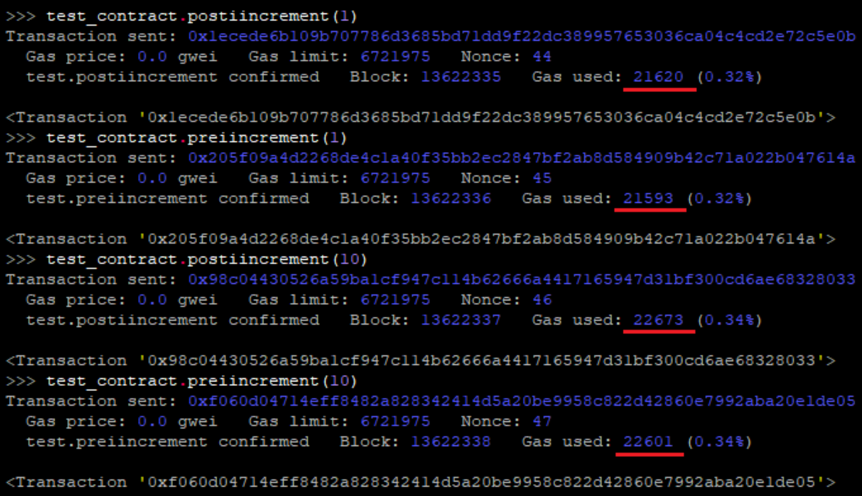
For example, based on the following test contract:

Listing 35: Test.sol

```
1 //SPDX-License-Identifier: MIT
2 pragma solidity 0.8.9;
```



```solidity

contract test {
    function postiincrement(uint256 iterations) public {
        for (uint256 i = 0; i < iterations; i++) {
        }
    }
    function preiincrement(uint256 iterations) public {
        for (uint256 i = 0; i < iterations; ++i) {
        }
    }
}
```



```
>>> test_contract.postiincrement(1)
Transaction sent: 0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 44
  test.postiincrement confirmed   Block: 13622335   Gas used: 21620 (0.32%)

<Transaction '0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b'>
>>> test_contract.preiincrement(1)
Transaction sent: 0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 45
  test.preiincrement confirmed   Block: 13622336   Gas used: 21593 (0.32%)

<Transaction '0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a'>
>>> test_contract.postiincrement(10)
Transaction sent: 0x98c04430526a59ba1cf947c114b62666a4417165947d31bf300cd6ae68328033
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 46
  test.postiincrement confirmed   Block: 13622337   Gas used: 22673 (0.34%)

<Transaction '0x98c04430526a59ba1cf947c114b62666a4417165947d31bf300cd6ae68328033'>
>>> test_contract.preiincrement(10)
Transaction sent: 0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 47
  test.preiincrement confirmed   Block: 13622338   Gas used: 22601 (0.34%)

<Transaction '0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05'>
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use ++i instead of i++ to increment the value of an uint variable inside a loop. This is not applicable outside of loops.

Remediation Plan:

SOLVED: The Playground labs team solved the above issue in commit 35fb92524b83ff8197a7127f7c9819317ac7ea92. As a result, for loops now uses ++i.

3.15 (HAL-15) CACHE ARRAY LENGTH IN FOR LOOPS CAN SAVE GAS - INFORMATIONAL

Description:

Reading the length of the array at each iteration of the loop requires 6 gas (3 for mload and 3 to place memory_offset) onto the stack. Caching the length of the array on the stack saves about 3 gas per iteration.

Code Location:

Listing 36: Governance.sol (Lines 94,102)

```
88      function getWeights(WeightSources memory weightSources)
89          public
90          view
91          returns (uint256 weightKAP, uint256 weightLP)
92      {
93          // Calc KAP voting weight
94          for (uint256 i = 0; i < weightSources.kapSources.length; i
    ↳ ++) {
95              if (weightSources.kapSources[i]) {
96                  weightKAP += IKAPSource(weightSourcesKAP[i]).
    ↳ weightKAP(
97                      msg.sender
98                  );
99              }
100         }
101         // Calc LP voting weight
102         for (uint256 i = 0; i < weightSources.lpSources.length; i
    ↳ ++) {
103             if (weightSources.lpSources[i]) {
104                 weightLP += ILPSource(weightSourcesLP[i]).weightLP
    ↳ (msg.sender);
105             }
106         }
107     }
```

Listing 37: Transactor.sol (Line 37)

```
23     function _transact(
24         address[] memory targets,
25         uint256[] memory values,
26         bytes[] memory data,
27         bool[] memory isDelegateCall
28     ) internal {
29         require(targets.length > 0, "Invalid array length");
30         require(targets.length == values.length, "Array length mismatch");
31         require(targets.length == data.length, "Array length mismatch");
32         require(
33             targets.length == isDelegateCall.length,
34             "Array length mismatch"
35         );
36
37         for (uint256 i = 0; i < targets.length; i++) {
38             if (isDelegateCall[i]) {
39                 Address.functionDelegateCall(targets[i], data[i]);
40             } else {
41                 Address.functionCallWithValue(targets[i], data[i], values[i]);
42             }
43         }
44     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider caching the length of the array. The example code can be seen below.

Listing 38: Updated Governance.sol (Lines 94,95,103,104)

```
88      function getWeights(WeightSources memory weightSources)
89          public
90          view
91          returns (uint256 weightKAP, uint256 weightLP)
92      {
93          // Calc KAP voting weight
94          uint256 kplength = weightSources.kapSources.length;
95          for (uint256 i = 0; i < kplength; i++) {
96              if (weightSources.kapSources[i]) {
97                  weightKAP += IKAPSource(weightSourcesKAP[i]).
 ↳ weightKAP(
98                      msg.sender
99                  );
100             }
101         }
102         // Calc LP voting weight
103         uint256 lplength = weightSources.lpSources.length;
104         for (uint256 i = 0; i < weightSources.lpSources.length; i
 ↳ ++) {
105             if (weightSources.lpSources[i]) {
106                 weightLP += ILPSource(weightSourcesLP[i]).weightLP
 ↳ (msg.sender);
107             }
108         }
109     }
```

Remediation Plan:

SOLVED: The Playground labs team solved the above issue in commit 35fb92524b83ff8197a7127f7c9819317ac7ea92. As a result, the contract cache the array length.

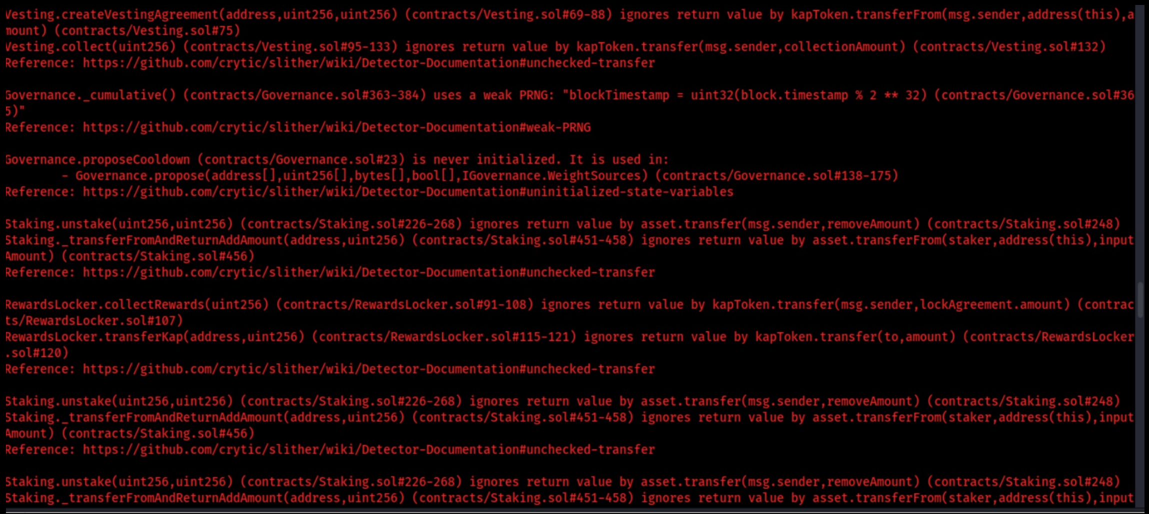
AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

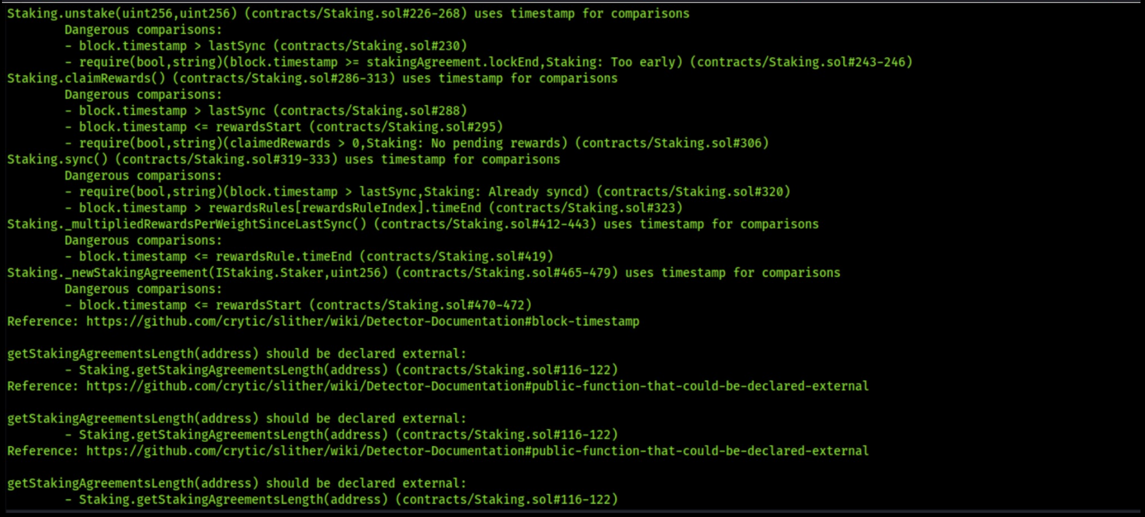
Description:

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

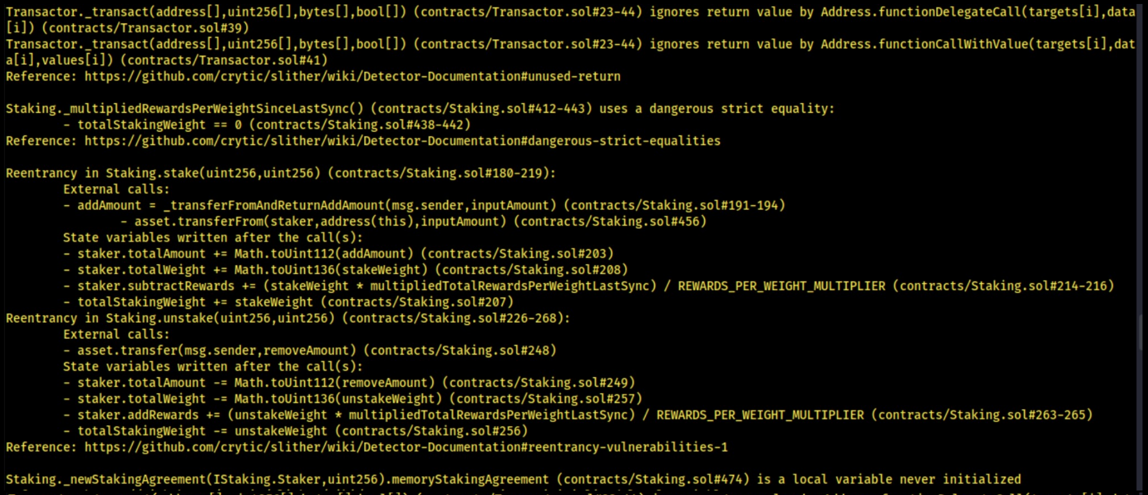
Results:



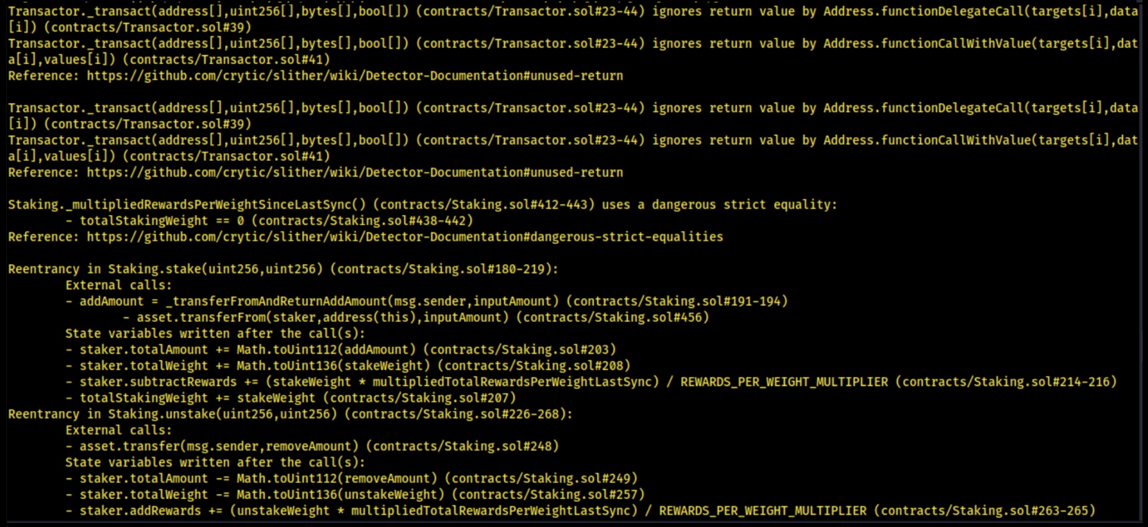
```
Vesting.createVestingAgreement(address,uint256,uint256) (contracts/Vesting.sol#69-88) ignores return value by kapToken.transferFrom(msg.sender,address(this),a
mount) (contracts/Vesting.sol#75)
Vesting.collect(uint256) (contracts/Vesting.sol#95-133) ignores return value by kapToken.transfer(msg.sender,collectionAmount) (contracts/Vesting.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

Governance._cumulative() (contracts/Governance.sol#363-384) uses a weak PRNG: "blockTimestamp = uint32(block.timestamp % 2 ** 32) (contracts/Governance.sol#36
5)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG

Governance.proposeCooldown (contracts/Governance.sol#23) is never initialized. It is used in:
        - Governance.propose(address[],uint256[],bytes[],bool[],IGovernance.WeightSources) (contracts/Governance.sol#138-175)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables

Staking.unstake(uint256,uint256) (contracts/Staking.sol#226-268) ignores return value by asset.transfer(msg.sender,removeAmount) (contracts/Staking.sol#248)
Staking._transferFromAndReturnAddAmount(address,uint256) (contracts/Staking.sol#451-458) ignores return value by asset.transferFrom(staker,address(this),input
Amount) (contracts/Staking.sol#456)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

RewardsLocker.collectRewards(uint256) (contracts/RewardsLocker.sol#91-108) ignores return value by kapToken.transfer(msg.sender,lockAgreement.amount) (contrac
ts/RewardsLocker.sol#107)
RewardsLocker.transferKap(address,uint256) (contracts/RewardsLocker.sol#115-121) ignores return value by kapToken.transfer(to,amount) (contracts/RewardsLocker
.sol#120)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

Staking.unstake(uint256,uint256) (contracts/Staking.sol#226-268) ignores return value by asset.transfer(msg.sender,removeAmount) (contracts/Staking.sol#248)
Staking._transferFromAndReturnAddAmount(address,uint256) (contracts/Staking.sol#451-458) ignores return value by asset.transferFrom(staker,address(this),input
Amount) (contracts/Staking.sol#456)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

Staking.unstake(uint256,uint256) (contracts/Staking.sol#226-268) ignores return value by asset.transfer(msg.sender,removeAmount) (contracts/Staking.sol#248)
Staking._transferFromAndReturnAddAmount(address,uint256) (contracts/Staking.sol#451-458) ignores return value by asset.transferFrom(staker,address(this),input
```



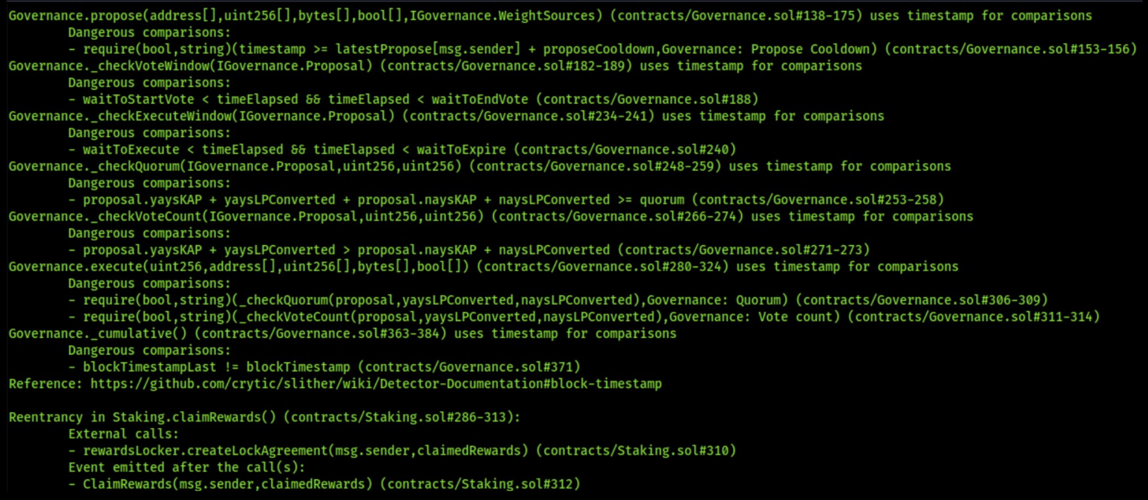
```
Staking.unstake(uint256,uint256) (contracts/Staking.sol#226-268) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > lastSync (contracts/Staking.sol#230)
        - require(bool,string)(block.timestamp >= stakingAgreement.lockEnd,Staking: Too early) (contracts/Staking.sol#243-246)
Staking.claimRewards() (contracts/Staking.sol#286-313) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > lastSync (contracts/Staking.sol#288)
        - block.timestamp <= rewardsStart (contracts/Staking.sol#295)
        - require(bool,string)(claimedRewards > 0,Staking: No pending rewards) (contracts/Staking.sol#306)
Staking.sync() (contracts/Staking.sol#319-333) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > lastSync,Staking: Already syncd) (contracts/Staking.sol#320)
        - block.timestamp > rewardsRules[rewardsRuleIndex].timeEnd (contracts/Staking.sol#323)
Staking._multipliedRewardsPerWeightSinceLastSync() (contracts/Staking.sol#412-443) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp <= rewardsRule.timeEnd (contracts/Staking.sol#419)
Staking._newStakingAgreement(IStaking.Staker,uint256) (contracts/Staking.sol#465-479) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp <= rewardsStart (contracts/Staking.sol#470-472)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

getStakingAgreementsLength(address) should be declared external:
        - Staking.getStakingAgreementsLength(address) (contracts/Staking.sol#116-122)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external

getStakingAgreementsLength(address) should be declared external:
        - Staking.getStakingAgreementsLength(address) (contracts/Staking.sol#116-122)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external

getStakingAgreementsLength(address) should be declared external:
        - Staking.getStakingAgreementsLength(address) (contracts/Staking.sol#116-122)
```



```
Transactor._transact(address[],uint256[],bytes[],bool[]) (contracts/Transactor.sol#23-44) ignores return value by Address.functionDelegateCall(targets[i],data
[i]) (contracts/Transactor.sol#39)
Transactor._transact(address[],uint256[],bytes[],bool[]) (contracts/Transactor.sol#23-44) ignores return value by Address.functionCallWithValue(targets[i],dat
a[i],values[i]) (contracts/Transactor.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Staking._multipliedRewardsPerWeightSinceLastSync() (contracts/Staking.sol#412-443) uses a dangerous strict equality:
        - totalStakingWeight == 0 (contracts/Staking.sol#438-442)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in Staking.stake(uint256,uint256) (contracts/Staking.sol#180-219):
        External calls:
        - addAmount = _transferFromAndReturnAddAmount(msg.sender,inputAmount) (contracts/Staking.sol#191-194)
                - asset.transferFrom(staker,address(this),inputAmount) (contracts/Staking.sol#456)
        State variables written after the call(s):
        - staker.totalAmount += Math.toUint112(addAmount) (contracts/Staking.sol#203)
        - staker.totalWeight += Math.toUint136(stakeWeight) (contracts/Staking.sol#208)
        - staker.subtractRewards += (stakeWeight * multipliedTotalRewardsPerWeightLastSync) / REWARDS_PER_WEIGHT_MULTIPLIER (contracts/Staking.sol#214-216)
        - totalStakingWeight += stakeWeight (contracts/Staking.sol#207)
Reentrancy in Staking.unstake(uint256,uint256) (contracts/Staking.sol#226-268):
        External calls:
        - asset.transfer(msg.sender,removeAmount) (contracts/Staking.sol#248)
        State variables written after the call(s):
        - staker.totalAmount -= Math.toUint112(removeAmount) (contracts/Staking.sol#249)
        - staker.totalWeight -= Math.toUint136(unstakeWeight) (contracts/Staking.sol#257)
        - staker.addRewards += (unstakeWeight * multipliedTotalRewardsPerWeightLastSync) / REWARDS_PER_WEIGHT_MULTIPLIER (contracts/Staking.sol#263-265)
        - totalStakingWeight -= unstakeWeight (contracts/Staking.sol#256)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

Staking._newStakingAgreement(IStaking.Staker,uint256).memoryStakingAgreement (contracts/Staking.sol#474) is a local variable never initialized
```



```
Transactor._transact(address[],uint256[],bytes[],bool[]) (contracts/Transactor.sol#23-44) ignores return value by Address.functionDelegateCall(targets[i],data
[i]) (contracts/Transactor.sol#39)
Transactor._transact(address[],uint256[],bytes[],bool[]) (contracts/Transactor.sol#23-44) ignores return value by Address.functionCallWithValue(targets[i],dat
a[i],values[i]) (contracts/Transactor.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Transactor._transact(address[],uint256[],bytes[],bool[]) (contracts/Transactor.sol#23-44) ignores return value by Address.functionDelegateCall(targets[i],data
[i]) (contracts/Transactor.sol#39)
Transactor._transact(address[],uint256[],bytes[],bool[]) (contracts/Transactor.sol#23-44) ignores return value by Address.functionCallWithValue(targets[i],dat
a[i],values[i]) (contracts/Transactor.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Staking._multipliedRewardsPerWeightSinceLastSync() (contracts/Staking.sol#412-443) uses a dangerous strict equality:
        - totalStakingWeight == 0 (contracts/Staking.sol#438-442)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in Staking.stake(uint256,uint256) (contracts/Staking.sol#180-219):
        External calls:
        - addAmount = _transferFromAndReturnAddAmount(msg.sender,inputAmount) (contracts/Staking.sol#191-194)
                - asset.transferFrom(staker,address(this),inputAmount) (contracts/Staking.sol#456)
        State variables written after the call(s):
        - staker.totalAmount += Math.toUint112(addAmount) (contracts/Staking.sol#203)
        - staker.totalWeight += Math.toUint136(stakeWeight) (contracts/Staking.sol#208)
        - staker.subtractRewards += (stakeWeight * multipliedTotalRewardsPerWeightLastSync) / REWARDS_PER_WEIGHT_MULTIPLIER (contracts/Staking.sol#214-216)
        - totalStakingWeight += stakeWeight (contracts/Staking.sol#207)
Reentrancy in Staking.unstake(uint256,uint256) (contracts/Staking.sol#226-268):
        External calls:
        - asset.transfer(msg.sender,removeAmount) (contracts/Staking.sol#248)
        State variables written after the call(s):
        - staker.totalAmount -= Math.toUint112(removeAmount) (contracts/Staking.sol#249)
        - staker.totalWeight -= Math.toUint136(unstakeWeight) (contracts/Staking.sol#257)
        - staker.addRewards += (unstakeWeight * multipliedTotalRewardsPerWeightLastSync) / REWARDS_PER_WEIGHT_MULTIPLIER (contracts/Staking.sol#263-265)
```



```
Governance.propose(address[],uint256[],bytes[],bool[],IGovernance.WeightSources) (contracts/Governance.sol#138-175) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(timestamp >= latestPropose[msg.sender] + proposeCooldown,Governance: Propose Cooldown) (contracts/Governance.sol#153-156)
Governance._checkVoteWindow(IGovernance.Proposal) (contracts/Governance.sol#182-189) uses timestamp for comparisons
        Dangerous comparisons:
        - waitToStartVote < timeElapsed && timeElapsed < waitToEndVote (contracts/Governance.sol#188)
Governance._checkExecuteWindow(IGovernance.Proposal) (contracts/Governance.sol#234-241) uses timestamp for comparisons
        Dangerous comparisons:
        - waitToExecute < timeElapsed && timeElapsed < waitToExpire (contracts/Governance.sol#240)
Governance._checkQuorum(IGovernance.Proposal,uint256,uint256) (contracts/Governance.sol#248-259) uses timestamp for comparisons
        Dangerous comparisons:
        - proposal.yaysKAP + yaysLPConverted + proposal.naysKAP + naysLPConverted >= quorum (contracts/Governance.sol#253-258)
Governance._checkVoteCount(IGovernance.Proposal,uint256,uint256) (contracts/Governance.sol#266-274) uses timestamp for comparisons
        Dangerous comparisons:
        - proposal.yaysKAP + yaysLPConverted > proposal.naysKAP + naysLPConverted (contracts/Governance.sol#271-273)
Governance.execute(uint256,address[],uint256[],bytes[],bool[]) (contracts/Governance.sol#280-324) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_checkQuorum(proposal,yaysLPConverted,naysLPConverted),Governance: Quorum) (contracts/Governance.sol#306-309)
        - require(bool,string)(_checkVoteCount(proposal,yaysLPConverted,naysLPConverted),Governance: Vote count) (contracts/Governance.sol#311-314)
Governance._cumulative() (contracts/Governance.sol#363-384) uses timestamp for comparisons
        Dangerous comparisons:
        - blockTimestampLast != blockTimestamp (contracts/Governance.sol#371)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Reentrancy in Staking.claimRewards() (contracts/Staking.sol#286-313):
        External calls:
        - rewardsLocker.createLockAgreement(msg.sender,claimedRewards) (contracts/Staking.sol#310)
        Event emitted after the call(s):
        - ClaimRewards(msg.sender,claimedRewards) (contracts/Staking.sol#312)
```

Based on the test results, some findings found by these tools were considered false positives, while some of these findings were real security concerns. All relevant findings were reviewed by the auditors and relevant findings were addressed in the report as security concerns.

4.2 AUTOMATED SECURITY SCAN

Description:

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

Results:

No relevant valid findings were found.

THANK YOU FOR CHOOSING

HALBORN