// HALBORN

KSMstarter Contract

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **September 9th, 2021 - September 25th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	09/20/2021	Juned Ansari
0.9	Document Updates	09/22/2021	Juned Ansari
1.0	Final Draft	09/27/2021	Gabi Urrutia
1.1	Remediation Plan	09/29/2021	Juned Ansari
1.1	Remediation Plan Review	09/29/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Juned Ansari	Halborn	Juned.Ansari@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

KSMstarter engaged Halborn to conduct a security assessment on their ksm-contracts smart contracts beginning on September 9th, 2021 and ending September 25th, 2021. This security assessment was scoped to the ksm-contracts smart contracts code in Solidity.

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

1.2 AUDIT SUMMARY

The team at Halborn was provided a week 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 to achieve the following:

- Ensure that all Nameless Contract functions are intended.
- Identify potential security issues with the assets in scope.

In summary, Halborn identified several security risk that were mostly addressed by KSMstarter 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 KSMstarter 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. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.

2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

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

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

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 : ksm-contracts github repository
The security assessment was scoped to the following smart contract:

Listing 1: ksm-contracts

```
1 KST.sol
2 VestingTeam.sol
3 VestingMarketing.sol
4 KSTStaking.sol
5 KSMIDO.sol
6 VestingCommunityIncentives.sol
7 TestERC20.sol
8 IDO.sol
9 IKSTStaking.sol
10 VestingTreasury.sol
11 KSMStarterLotteryRNG.sol
12 IDOFactory.sol
13 USDT.sol
```

OUT-OF-SCOPE : External libraries and economics attacks
Fixed-Commit-ID : 4feddbc179f8a511f7d3bf0b076843cde114912e, 8307469d2121019c63915f35923db04d854f339d

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	5	4

LIKELIHOOD

IMPACT

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

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - MISSING RE-ENTRANCY PROTECTION	Medium	SOLVED - 09/28/2021
HAL02 - UNCHECKED TRANSFER	Medium	SOLVED - 09/28/2021
HAL03 - MISSING ZERO-ADDRESS CHECK	Low	SOLVED - 09/28/2021
HAL04 - INCOMPATIBILITY WITH INFLATIONARY TOKENS	Low	SOLVED - 09/28/2021
HAL05 - USE OF BLOCK-TIMESTAMP	Low	RISK ACCEPTED
HAL06 - FLOATING PRAGMA	Low	SOLVED - 09/28/2021
HAL07 - OUTDATED DEPENDENCIES	Low	SOLVED - 09/28/2021
HAL08 - PRAGMA TOO RECENT	Informational	RISK ACCEPTED
HAL09 - MISSING EVENTS EMITTING	Informational	SOLVED - 09/28/2021
HAL10 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED - 09/28/2021
HAL11 - STATE VARIABLE SHADOWING	Informational	SOLVED - 09/28/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) MISSING RE-ENTRANCY PROTECTION - MEDIUM

Description:

It was identified that KSMStarter Contracts are missing nonReentrant guard. In contract IDO.sol, functions register and in contract KSTStaking .sol, functions stake and unstake.sol, and in contract KSMIDO.sol function buy(uint256) are missing nonReentrant guard. Also, in these functions, external calls are called before all state changes are resolved, making it vulnerable to a Reentrancy attack.

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 it's 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 2: IDO.sol (Lines 215,216,217)

```
215         staking.lock(msg.sender, claimStartTime); // locks users
                staked balance
216         idoParticipants = idoParticipants.add(1);
217         emit Registered(msg.sender, claimStartTime, lockedBal,
                tier);
218     }
```

Listing 3: KSTStaking.sol (Lines 45,47,48)

```
43      function stake(uint256 value) external notHalted {
44          require(value > 0, "KSTStaking: stake value should be
                greater than 0");
45          _token.safeTransferFrom(_msgSender(), address(this), value
                );
46
```

```
47          _balances[_msgSender()] = _balances[_msgSender()].add(
                value);
48          emit Stake(_msgSender(),block.timestamp,value);
49      }
```

Listing 4: KSTStaking.sol (Lines 55,56)

```
51      function unstake(uint256 value) external lockable {
52          require(_balances[_msgSender()] >= value, 'KSTStaking:
                insufficient staked balance');
53
54          _balances[_msgSender()] = _balances[_msgSender()]-value;
55          _token.safeTransfer(_msgSender(), value);
56          emit Unstake(_msgSender(),block.timestamp,value);
57      }
```

Listing 5: KSMIDO.sol (Lines 53,64,65,66,67,68)

```
53 paymentToken.safeTransferFrom(msg.sender, address(this),
       _paymentAmount);
54          uint256 tokensToPurchase = _paymentAmount.mul(tokenPrice);
55          uint256 difference;
56          if (_paymentAmount.add(totalRaised) > raiseCap) {
57              difference = raiseCap.sub(_paymentAmount); //fill the
                    remaining cap
58              tokensToPurchase = difference.mul(tokenPrice);
59          } else {
60              difference = _paymentAmount;
61          }
62          uint256 userEthAmount = user.tokenAmount.div(tokenPrice);
                // full amount
63          require(difference <= userEthAmount, "not correct eth
                value");
64          user.claimAmount = tokensToPurchase; // set how much
                tokens can claim
65          user.swapAmount = difference; // set how much movr was
                sent
66          user.tokenAmount = difference.mul(tokenPrice); // set
                token amount based on eth sent
67          totalRaised = totalRaised + difference; // increase total
                raised
68          emit Bought(msg.sender, difference);
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

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

Remediation Plan:

SOLVED: The KSM-Starter team solved the issue by adding the nonReentrant modifier.

3.2 (HAL-02) UNCHECKED TRANSFER - MEDIUM

Description:

In contract KSMStarterLotteryRNG.sol the return value of an external transfer call LINK.transfer(msg.sender,LINK.balanceOf(address(this))) is not checked. It should be noted that token do not revert in case of failure and return false. If one of these tokens is used, a deposit would not revert if the transfer fails, and an attacker could deposit tokens for free.

Code Location:

Listing 6: KSMStarterLotteryRNG.sol (Lines 56)

```
55     function returnLINK() external ownerOrOperator {
56         LINK.transfer(msg.sender, LINK.balanceOf(address(this)));
57     }
58
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

It is recommended to use SafeERC20, or ensure that the transfer return value is checked. The ERC20 standard recommends throwing exceptions in functions transfer and transferFrom.

References:

ERC20 API: Transfer and TransferFrom

Remediation Plan:

SOLVED: The KSM-Starter team solved the issue by using SafeERC20 implementation. Also, they added the safetransfer function to the code:

Listing 7

```
1 IERC20(address(LINK)).safeTransfer(msg.sender, LINK.balanceOf(
      address(this)));
```

3.3 (HAL-03) MISSING ZERO-ADDRESS CHECK - LOW

Description:

There are multiple instances found where Address validation is missing. Lack of zero address validation has been found when assigning user supplied address values to state variables directly. In VestingCommunityIncentives.sol, VestingMarketing.sol, VestingTeam.sol and VestingTreasury.sol contracts function setBeneficiary(address) lacks a zero-check on _addy.

Code Location:

Listing 8: VestingCommunityIncentives.sol (Lines 81)

```
79     function setBeneficiary(address _addy) external {
80         require(msg.sender == beneficiary);
81         beneficiary = _addy;
82     }
```

Listing 9: VestingMarketing.sol (Lines 81)

```
79     function setBeneficiary(address _addy) external {
80         require(msg.sender == beneficiary);
81         beneficiary = _addy;
82     }
```

Listing 10: VestingTeam.sol (Lines 81)

```
79     function setBeneficiary(address _addy) external {
80         require(msg.sender == beneficiary);
81         beneficiary = _addy;
82     }
```

Listing 11: VestingTreasury.sol (Lines 81)

```
79      function setBeneficiary(address _addy) external {
80          require(msg.sender == beneficiary);
81          beneficiary = _addy;
82      }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Although administrative restrictions are imposed to this function due to the OpenZeppelin RBAC it is better to add proper address validation when assigning a value to a variable from user supplied inputs.

Remediation Plan:

SOLVED: KSM-Starter team solved the issue by adding the zero address check (`require(_addy != address(0));`) to the code.

3.4 (HAL-04) INCOMPATIBILITY WITH INFLATIONARY TOKENS - LOW

Description:

In multiple functions KSMStarter uses OpenZeppelin's safeTransferFrom and safeTransfer to handle the token transfers. These functions call transferFrom and transfer internally in the token contract to actually execute the transfer. However, since the actual amount transferred i.e. the delta of previous (before transfer) and current (after transfer) balance is not verified, a malicious user may list a custom ERC20 token with the transferFrom or transfer function modified in such a way that it e.g. does not transfer any tokens at all and the attacker is still going to have their liquidity pool tokens minted anyway. In this case both tokens are set in the constructor by the creator of the contract, so they are trusted, but it would be still a good practice to perform this check.

Code Location:

Listing 12: IDO.sol

```
1 idoToken.safeTransfer(msg.sender, claimable.idoTokenAmount);
2 swapToken.safeTransferFrom(msg.sender, address(this),
      swapTokenAmount);
3 swapToken.safeTransfer(owner(), swapToken.balanceOf(address(this))
      );
4 idoToken.safeTransfer(owner(), totalIdoTokens.sub(totalSwapped));
```

Listing 13: KSTStaking.sol

```
1 _token.safeTransferFrom(_msgSender(), address(this), value);
2 _token.safeTransfer(_msgSender(), value);
```

Listing 14: VestingCommunityIncentives.sol

```
1 kstToken.safeTransfer(beneficiary, RELEASEAMOUNT);
2 kstToken.safeTransfer(beneficiary, RELEASEAMOUNT.mul(multiplier));
3 kstToken.safeTransfer(beneficiary, kstToken.balanceOf(address(this
      )));
```

Listing 15: VestingTreasury.sol

```
1 kstToken.safeTransfer(beneficiary, RELEASEAMOUNT);
2 kstToken.safeTransfer(beneficiary, RELEASEAMOUNT.mul(multiplier));
3 kstToken.safeTransfer(beneficiary, kstToken.balanceOf(address(this
      )));
```

Listing 16: VestingMarketing.sol

```
1 kstToken.safeTransfer(beneficiary, RELEASEAMOUNT);
2 kstToken.safeTransfer(beneficiary, RELEASEAMOUNT.mul(multiplier));
3 kstToken.safeTransfer(beneficiary, kstToken.balanceOf(address(this
      )));
```

Listing 17: VestingTeam.sol

```
1 kstToken.safeTransfer(beneficiary, RELEASEAMOUNT);
2 kstToken.safeTransfer(beneficiary, RELEASEAMOUNT.mul(multiplier));
3 kstToken.safeTransfer(beneficiary, kstToken.balanceOf(address(this
      )));
```

Listing 18: KSMIDO.sol

```
1 paymentToken.safeTransferFrom(msg.sender, address(this),
      _paymentAmount);
2 idoToken.safeTransfer(msg.sender, sendAmount);
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendations:

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

Remediation Plan:

SOLVED: KSM-Starter team claims that tokens will always be ERC20 and validated on the client-side. Thus, only the owner is allowed to input these tokens in the smart contract.

3.5 (HAL-05) USE OF BLOCK-TIMESTAMP - LOW

Description:

During a manual review, the use of block.timestamp in IDO.sol, KSTStaking.sol, VestingCommunityIncentives.sol, VestingMarketing.sol, VestingTeam.sol, and VestingTreasury.sol were observed. The contract developers should be aware that this does not mean current time. now is an alias for block.timestamp. The value of block.timestamp can be influenced by miners to a certain degree, so the testers should be warned that this may have some risk if miners collude on time manipulation to influence the price oracles. Miners can influence the timestamp by a tolerance of 900 seconds.

Code Location:

Listing 19: IDO.sol (Lines 239,240,241,242,243)

```
239             require(
240                 block.timestamp >= t1.swapStart &&
241                     block.timestamp <= t1.swapEnd,
242                 "IDO: t1 swap start/end not time"
243             );
244             require(swap(swapTokenAmount), "IDO: t1 swap error.");
```

Listing 20: IDO.sol (Lines 249,250,251,252,253)

```
249             require(
250                 block.timestamp >= t2.swapStart &&
251                     block.timestamp <= t2.swapEnd,
252                 "IDO: t2 swap start/end not time"
253             );
254             require(
```

Listing 21: IDO.sol (Lines 267,268,269,270,271)

```
267             require(
268                 block.timestamp >= t3.swapStart &&
269                     block.timestamp <= t3.swapEnd,
270                 "IDO: t3 swap start/end not time"
271             );
272             require(swap(swapTokenAmount), "IDO: t3 swap error.");
```

Listing 22: IDO.sol (Lines 277,278,279,280,281)

```
277             require(
278                 block.timestamp >= t4.swapStart &&
279                     block.timestamp <= t4.swapEnd,
280                 "IDO: t4 swap start/end not time"
281             );
282             require(
```

Listing 23: IDO.sol (Lines 291,292,293,294,295)

```
291             require(
292                 block.timestamp >= t5.swapStart &&
293                     block.timestamp <= t5.swapEnd,
294                 "IDO: t5 swap start/end not time"
295             );
296             require(
```

Listing 24: IDO.sol (Lines 310)

```
310         require(block.timestamp >= claimStartTime, "IDO: Claim
               start not yet.");
311         require(claimable.claimedAmount == 0, "IDO: Already
               claimed");
```

Listing 25: IDO.sol (Lines 452)

```
452         require(block.timestamp >= t5.swapEnd, "IDO: Swap time not
                done yet");
453         uint256 totalSwapped = getTotalSwapped();
```

Listing 26: KSTStaking.sol (Lines 60)

```
60         require(userUnlockTime > block.timestamp, "KSTStaking:
               unlock is in the past");
61         if (_unlockTime[user] < userUnlockTime) {
```

Listing 27: VestingCommunityIncentives.sol (Lines 36,37,38,39,40,41)

```
36         require(block.timestamp >= startTime, "start time not set"
               );
37         require(
38             block.timestamp >= lastClaimTime.add(CLAIM_DELAY),
39             "delay since last claim not passed"
40         );
41         if (block.timestamp > END_DAY) {
42             claimDust();
```

Listing 28: VestingCommunityIncentives.sol (Lines 45)

```
45             if (currentEpoch > epoch) {
46                 uint256 multiplier = currentEpoch.sub(epoch);
```

Listing 29: VestingCommunityIncentives.sol (Lines 63)

```
63         if (currentEpoch > epoch) {
64             uint256 multiplier = currentEpoch.sub(epoch);
```

Listing 30: VestingCommunityIncentives.sol (Lines 72)

```
72         require(block.timestamp >= END_DAY, "Vesting not yet
               finished");
73         kstToken.safeTransfer(
```

Listing 31: VestingMarketing.sol (Lines 36,37,38,39,40,41)

```
36         require(block.timestamp >= startTime, "start time not set"
               );
37         require(
38             block.timestamp >= lastClaimTime.add(CLAIM_DELAY),
39             "delay since last claim not passed"
```

```
40          );
41          if (block.timestamp > END_DAY) {
42              claimDust();
```

Listing 32: VestingMarketing.sol (Lines 45)

```
45              if (currentEpoch > epoch) {
46                  uint256 multiplier = currentEpoch.sub(epoch);
```

Listing 33: VestingMarketing.sol (Lines 63)

```
63          if (currentEpoch > epoch) {
64              uint256 multiplier = currentEpoch.sub(epoch);
```

Listing 34: VestingMarketing.sol (Lines 72)

```
72          require(block.timestamp >= END_DAY, "Vesting not yet
                finished");
73          kstToken.safeTransfer(
```

Listing 35: VestingTeam.sol (Lines 36,37,38,39,40,41)

```
36          require(block.timestamp >= startTime, "start time not set"
                );
37          require(
38              block.timestamp >= lastClaimTime.add(CLAIM_DELAY),
39              "delay since last claim not passed"
40          );
41          if (block.timestamp > END_DAY) {
42              claimDust();
```

Listing 36: VestingTeam.sol (Lines 45)

```
45              if (currentEpoch > epoch) {
46                  uint256 multiplier = currentEpoch.sub(epoch);
```

Listing 37: VestingTeam.sol (Lines 63)

```
63          if (currentEpoch > epoch) {
64              uint256 multiplier = currentEpoch.sub(epoch);
```

Listing 38: VestingTeam.sol (Lines 72)

```
72          require(block.timestamp >= END_DAY, "Vesting not yet
                finished");
73          kstToken.safeTransfer(
```

Listing 39: VestingTreasury.sol (Lines 36,37,38,39,40,41)

```
36          require(block.timestamp >= startTime, "start time not set"
                );
37          require(
38              block.timestamp >= lastClaimTime.add(CLAIM_DELAY),
39              "delay since last claim not passed"
40          );
41          if (block.timestamp > END_DAY) {
42              claimDust();
```

Listing 40: VestingTreasury.sol (Lines 45)

```
45              if (currentEpoch > epoch) {
46                  uint256 multiplier = currentEpoch.sub(epoch);
```

Listing 41: VestingTreasury.sol (Lines 63)

```
63          if (currentEpoch > epoch) {
64              uint256 multiplier = currentEpoch.sub(epoch);
```

Listing 42: VestingTreasury.sol (Lines 72)

```
72          require(block.timestamp >= END_DAY, "Vesting not yet
                finished");
73          kstToken.safeTransfer(
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Use block.number instead of block.timestamp or now to reduce the risk of MEV attacks. Check if the timescale of the project occurs across years, days and months rather than seconds. If possible, it is recommended to use Oracles.

Remediation Plan:

RISK ACCEPTED: The KSM-Starter team accepts the risk and continues using block.timestamp as 900 seconds fluctuation is not very crucial to the client.

3.6 (HAL-06) FLOATING PRAGMA - LOW

Description:

KSM contract IDOFactory.sol, KST.sol, IDO.sol, KSMIDO.sol, KSTStaking.sol , IKSTStaking.sol, TestERC20.sol, KSMStarterLotteryRNG.sol, and USDT.sol uses the floating pragma ^0.8.6. Contract should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, either an outdated compiler version that might introduce bugs that affect the contract system negatively or a pragma version too new which has not been extensively tested.

Code Location:

Listing 43: (Lines 2)

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.6;
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendations:

Consider locking the pragma version with known bugs for the compiler version. When possible, do not use floating pragma in the final live deployment. Specifying a fixed compiler version ensures that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Remediation Plan:

SOLVED: KSM-Starter team locked the pragma version.

3.7 (HAL-07) OUTDATED DEPENDENCIES - LOW

Description:

It was noticed that the 4.1.0 version of openzepplin-contracts is used in smart contracts. However, the latest version of those libraries is 4.3.2, which fixes a vulnerability in UUPSUpgradeable.

Code Location:

Listing 44: package.json (Lines 27)

```
25   "dependencies": {
26     "@chainlink/contracts": "^0.2.0",
27     "@openzeppelin/contracts": "^4.1.0",
28     "@openzeppelin/test-helpers": "^0.5.11",
29     "dotenv": "^10.0.0"
30   }
31 }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Even though UUPSUpgradeable is not used directly within contracts, it is always important to keep all libraries up-to-date.

References:

Open Zeppelin Advisory
UUPS Implementation Workaround

Remediation Plan:

SOLVED: KSM-Starter team updated the library openzepplin-contracts version to 4.3.2.

3.8 (HAL-08) PRAGMA TOO RECENT - INFORMATIONAL

Description:

KSM contract uses one of the latest pragma version (0.8.6) which was released on June 22nd, 2021. The latest pragma version (0.8.7) was released in August 2021. Many pragma versions have been lately released, going from version 0.7.x to the recently released version 0.8.x. in just 6 months.

Reference: https://github.com/ethereum/solidity/releases

In the Solitidy Github repository, there is a json file where are all bugs finding in the different compiler versions. It should be noted that pragma 0.6.12 and 0.7.6 are widely used by Solidity developers and have been extensively tested in many security audits.

Reference: https://github.com/ethereum/solidity/blob/develop/docs/bugs_by_version.json

Code Location:

Listing 45: (Lines 2)

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.6;
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

If possible, consider using the latest stable pragma version that has been thoroughly tested to prevent potential undiscovered vulnerabilities such as pragma between 0.6.12 - 0.7.6.

Remediation Plan:

Risk-Accepted: The KSM-Starter team accepts the risk and continues using pragma version 0.8.6.

3.9 (HAL-09) MISSING EVENTS EMITTING - INFORMATIONAL

Description:

It has been observed that important functionality is missing emitting event for some functions on the IDO.sol contract. These functions should emit events. Events are a method of informing the transaction initiator about the actions taken by the called function. It logs its emitted parameters in a specific log history, which can be accessed outside of the contract using some filter parameters. These functions should emit events.

Code Location:

Listing 46: IDO.sol (Lines 443)

```
440     function setSwapPrice(uint256 _price) external ownerOrOperator
            {
441         require(!priceSet, "IDO: price already set.");
442         priceSet = true;
443         swapPrice = _price;
444     }
```

Listing 47: IDO.sol (Lines 461)

```
459     function setOperator(address _operator) public ownerOrOperator
            {
460         require(_operator != address(0));
461         operator = _operator;
462     }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

For best security practices, consider as much as possible declaring events at the end of the function. Events can be used to detect the end of the operation.

Remediation Plan:

SOLVED: KSM-Starter team added events to the above function.

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

Description:

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

Code Location:

Below are smart contracts and their corresponding functions affected:

IDO.sol:
claimTokens() createIDO(uint256[15],address,address,address,uint256) getTier1Registrants() getTotalRaised() register() setOperator(address) swapTokens()

IDOFactory.sol:
createIDO(uint256[15],address,address,address,uint256) setOperator(address)

KSMStarterLotteryRNG.sol:
expand(uint256,uint256)

VestingCommunityIncentives.sol:
getClaimAmount()

VestingMarketing.sol:
getClaimAmount()

VestingTeam.sol:

getClaimAmount()

VestingTreasury.sol:
getClaimAmount()

KSMIDO.sol:
buy(uint256) claim() updateUser(address,uint256)

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider as much as possible declaring external variables instead of public variables. As for best practice, you should use external if you expect that the function will only be called externally and use public if you need to call the function internally. To sum up, all can access to public functions, external functions only can be accessed externally and internal functions can only be called within the contract.

Remediation Plan:

SOLVED: KSM-Starter team updated the code and declared external functions instead of public.

3.11 (HAL-11) STATE VARIABLE SHADOWING - INFORMATIONAL

Description:

Contract KSMStarterLotteryRNG.sol function requestRandomNumbers() state variable requestId shadows requestId of KSMStarterLotteryRNG.sol. Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Code Location:

Listing 48: KSMStarterLotteryRNG.sol (Lines 9)

```
9     bytes32 public requestId;
```

Listing 49: KSMStarterLotteryRNG.sol (Lines 50)

```
50        returns (bytes32 requestId)
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

It is recommended to carefully review the variable storage design in the contracts to avoid possible ambiguities. Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables. Though all these state variables are meant to be internal, it is best practice to set the visibility.

Remediation Plan:

SOLVED: KSM-Starter team solved the issue.

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:

```
IDO.calcSwapAmount(IDO.Tiers) (contracts/IDO.sol#331-397) performs a multiplication on the result of a division:
        -userPerc = t1.participant[msg.sender].mul(100000).div(t1.totalLocked) (contracts/IDO.sol#343-345)
        -idoTokenAmount = tierAlloc.mul(userPerc).div(100000) (contracts/IDO.sol#348)
        -tierAlloc = totalIdoTokens.mul(t3.perc).div(1000) (contracts/IDO.sol#364)
IDO.calcSwapAmount(IDO.Tiers) (contracts/IDO.sol#331-397) performs a multiplication on the result of a division:
        -idoTokenAmount = tierAlloc.mul(userPerc).div(100000) (contracts/IDO.sol#359)
        -userPerc = t3.participant[msg.sender].mul(100000).div(t3.totalLocked) (contracts/IDO.sol#365-367)
        -tierAlloc = totalIdoTokens.mul(t5.perc).div(1000) (contracts/IDO.sol#386)
IDO.calcSwapAmount(IDO.Tiers) (contracts/IDO.sol#331-397) performs a multiplication on the result of a division:
        -idoTokenAmount = tierAlloc.mul(userPerc).div(100000) (contracts/IDO.sol#370)
        -tierAlloc = totalIdoTokens.mul(t4.perc).div(1000) (contracts/IDO.sol#375)
        -userPerc = t5.participant[msg.sender].mul(100000).div(t5.totalLocked) (contracts/IDO.sol#387-389)
IDO.calcSwapAmount(IDO.Tiers) (contracts/IDO.sol#331-397) performs a multiplication on the result of a division:
        -tierAlloc = totalIdoTokens.mul(t4.perc).div(1000) (contracts/IDO.sol#375)
        -userPerc = t4.participant[msg.sender].mul(100000).div(t4.totalLocked) (contracts/IDO.sol#376-378)
        -idoTokenAmount = tierAlloc.mul(userPerc).div(100000) (contracts/IDO.sol#392)
IDO.calcSwapAmount(IDO.Tiers) (contracts/IDO.sol#331-397) performs a multiplication on the result of a division:
        -tierAlloc = totalIdoTokens.mul(t4.perc).div(1000) (contracts/IDO.sol#375)
        -userPerc = t4.participant[msg.sender].mul(100000).div(t4.totalLocked) (contracts/IDO.sol#376-378)
        -idoTokenAmount = tierAlloc.mul(userPerc).div(100000) (contracts/IDO.sol#381)
Reentrancy in VestingTreasury.claimTokens() (contracts/VestingTreasury.sol#35-59):
        External calls:
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT.mul(multiplier)) (contracts/VestingTreasury.sol#47-50)
        State variables written after the call(s):
        - epoch = epoch.add(multiplier) (contracts/VestingTreasury.sol#51)
Reentrancy in VestingTreasury.claimTokens() (contracts/VestingTreasury.sol#35-59):
        External calls:
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT) (contracts/VestingTreasury.sol#53)
        State variables written after the call(s):
        - epoch = epoch.add(1) (contracts/VestingTreasury.sol#54)
Reentrancy in VestingTreasury.claimTokens() (contracts/VestingTreasury.sol#35-59):
        External calls:
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT.mul(multiplier)) (contracts/VestingTreasury.sol#47-50)
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT) (contracts/VestingTreasury.sol#53)
        State variables written after the call(s):
        - lastClaimTime = block.timestamp (contracts/VestingTreasury.sol#56)
```

AUTOMATED TESTING

```
Reentrancy in VestingCommunityIncentives.claimTokens() (contracts/VestingCommunityIncentives.sol#35-59):
        External calls:
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT.mul(multiplier)) (contracts/VestingCommunityIncentives.sol#47-50)
        State variables written after the call(s):
        - epoch = epoch.add(multiplier) (contracts/VestingCommunityIncentives.sol#51)
Reentrancy in VestingCommunityIncentives.claimTokens() (contracts/VestingCommunityIncentives.sol#35-59):
        External calls:
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT) (contracts/VestingCommunityIncentives.sol#53)
        State variables written after the call(s):
        - epoch = epoch.add(1) (contracts/VestingCommunityIncentives.sol#54)
Reentrancy in VestingCommunityIncentives.claimTokens() (contracts/VestingCommunityIncentives.sol#35-59):
        External calls:
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT.mul(multiplier)) (contracts/VestingCommunityIncentives.sol#47-50)
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT) (contracts/VestingCommunityIncentives.sol#53)
        State variables written after the call(s):
        - lastClaimTime = block.timestamp (contracts/VestingCommunityIncentives.sol#56)
Reentrancy in VestingMarketing.claimTokens() (contracts/VestingMarketing.sol#35-59):
        External calls:
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT.mul(multiplier)) (contracts/VestingMarketing.sol#47-50)
        State variables written after the call(s):
        - epoch = epoch.add(multiplier) (contracts/VestingMarketing.sol#51)
Reentrancy in VestingMarketing.claimTokens() (contracts/VestingMarketing.sol#35-59):
        External calls:
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT) (contracts/VestingMarketing.sol#53)
        State variables written after the call(s):
        - epoch = epoch.add(1) (contracts/VestingMarketing.sol#54)
Reentrancy in VestingMarketing.claimTokens() (contracts/VestingMarketing.sol#35-59):
        External calls:
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT.mul(multiplier)) (contracts/VestingMarketing.sol#47-50)
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT) (contracts/VestingMarketing.sol#53)
        State variables written after the call(s):
        - lastClaimTime = block.timestamp (contracts/VestingMarketing.sol#56)
Reentrancy in VestingTeam.claimTokens() (contracts/VestingTeam.sol#35-59):
        External calls:
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT.mul(multiplier)) (contracts/VestingTeam.sol#47-50)
        State variables written after the call(s):
        - epoch = epoch.add(multiplier) (contracts/VestingTeam.sol#51)
Reentrancy in VestingTeam.claimTokens() (contracts/VestingTeam.sol#35-59):
        External calls:
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT) (contracts/VestingTeam.sol#53)
        State variables written after the call(s):
        - epoch = epoch.add(1) (contracts/VestingTeam.sol#54)
Reentrancy in VestingTeam.claimTokens() (contracts/VestingTeam.sol#35-59):
        External calls:
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT.mul(multiplier)) (contracts/VestingTeam.sol#47-50)
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT) (contracts/VestingTeam.sol#53)
        State variables written after the call(s):
        - lastClaimTime = block.timestamp (contracts/VestingTeam.sol#56)
Reentrancy in IDO.claimTokens() (contracts/IDO.sol#307-315):
        External calls:
        - idoToken.safeTransfer(msg.sender,claimable.idoTokenAmount) (contracts/IDO.sol#313)
        Event emitted after the call(s):
        - Claimed(msg.sender,claimable.idoTokenAmount) (contracts/IDO.sol#314)
Reentrancy in IDOFactory.createIDO(uint256[15],address,address,address,uint256) (contracts/IDOFactory.sol#11-26):
        External calls:
        - ido.setSwapPrice(_swapTokenPrice) (contracts/IDOFactory.sol#20)
        - ido.transferOwnership(msg.sender) (contracts/IDOFactory.sol#21)
        - staking.addIDO(address(ido)) (contracts/IDOFactory.sol#23)
        Event emitted after the call(s):
        - IDOCreated(address(ido)) (contracts/IDOFactory.sol#24)
Reentrancy in IDO.register() (contracts/IDO.sol#182-218):
        External calls:
        - staking.lock(msg.sender,claimStartTime) (contracts/IDO.sol#215)
        Event emitted after the call(s):
        - Registered(msg.sender,claimStartTime,lockedBal,tier) (contracts/IDO.sol#217)
Reentrancy in IDO.swap(uint256) (contracts/IDO.sol#317-323):
        External calls:
        - swapToken.safeTransferFrom(msg.sender,address(this),swapTokenAmount) (contracts/IDO.sol#320)
        Event emitted after the call(s):
        - Swapped(msg.sender,swapTokenAmount) (contracts/IDO.sol#321)
```

AUTOMATED TESTING

```
Reentrancy in IDO.register() (contracts/IDO.sol#182-218):
        External calls:
        - staking.lock(msg.sender,claimStartTime) (contracts/IDO.sol#215)
        State variables written after the call(s):
        - idoParticipants = idoParticipants.add(1) (contracts/IDO.sol#216)
Reentrancy in VRFConsumerBase.requestRandomness(bytes32,uint256) (chainlink/contracts/src/v0.8/VRFConsumerBase.sol#158-180):
       External calls:
       - LINK.transferAndCall(vrfCoordinator,_fee,abi.encode(_keyHash,USER_SEED_PLACEHOLDER)) (chainlink/contracts/src/v0.8/VRFConsumerBase.sol#167)
       State variables written after the call(s):
       - nonces[_keyHash] = nonces[_keyHash] + 1 (chainlink/contracts/src/v0.8/VRFConsumerBase.sol#178)
Reentrancy in VestingTreasury.claimTokens() (contracts/VestingTreasury.sol#35-59):
        External calls:
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT.mul(multiplier)) (contracts/VestingTreasury.sol#47-50)
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT) (contracts/VestingTreasury.sol#53)
        Event emitted after the call(s):
        - Claimed(RELEASEAMOUNT,epoch) (contracts/VestingTreasury.sol#57)
VestingCommunityIncentives.claimTokens() (contracts/VestingCommunityIncentives.sol#35-59) uses timestamp for comparisons
       Dangerous comparisons:
       - require(bool,string)(block.timestamp >= startTime,start time not set) (contracts/VestingCommunityIncentives.sol#36)
       - require(bool,string)(block.timestamp >= lastClaimTime.add(CLAIM_DELAY),delay since last claim not passed) (contracts/VestingCommunityIncentives.sol#37
-40)
       - block.timestamp > END_DAY (contracts/VestingCommunityIncentives.sol#41)
       - currentEpoch > epoch (contracts/VestingCommunityIncentives.sol#45)
VestingCommunityIncentives.getClaimAmount() (contracts/VestingCommunityIncentives.sol#61-69) uses timestamp for comparisons
       Dangerous comparisons:
       - currentEpoch > epoch (contracts/VestingCommunityIncentives.sol#63)
VestingCommunityIncentives.claimDust() (contracts/VestingCommunityIncentives.sol#71-77) uses timestamp for comparisons
       Dangerous comparisons:
       - require(bool,string)(block.timestamp >= END_DAY,Vesting not yet finished) (contracts/VestingCommunityIncentives.sol#72)
Reentrancy in VestingCommunityIncentives.claimTokens() (contracts/VestingCommunityIncentives.sol#35-59):
        External calls:
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT.mul(multiplier)) (contracts/VestingCommunityIncentives.sol#47-50)
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT) (contracts/VestingCommunityIncentives.sol#53)
        Event emitted after the call(s):
        - Claimed(RELEASEAMOUNT,epoch) (contracts/VestingCommunityIncentives.sol#57)
Reentrancy in KSTStaking.stake(uint256) (contracts/KSTStaking.sol#43-49):
        External calls:
        - _token.safeTransferFrom(_msgSender(),address(this),value) (contracts/KSTStaking.sol#45)
        Event emitted after the call(s):
        - Stake(_msgSender(),block.timestamp,value) (contracts/KSTStaking.sol#48)
Reentrancy in KSTStaking.unstake(uint256) (contracts/KSTStaking.sol#51-57):
        External calls:
        - _token.safeTransfer(_msgSender(),value) (contracts/KSTStaking.sol#55)
        Event emitted after the call(s):
        - Unstake(_msgSender(),block.timestamp,value) (contracts/KSTStaking.sol#56)
Reentrancy in KSTStaking.stake(uint256) (contracts/KSTStaking.sol#43-49):
        External calls:
        - _token.safeTransferFrom(_msgSender(),address(this),value) (contracts/KSTStaking.sol#45)
        State variables written after the call(s):
        - _balances[_msgSender()] = _balances[_msgSender()].add(value) (contracts/KSTStaking.sol#47)
Reentrancy in VestingMarketing.claimTokens() (contracts/VestingMarketing.sol#35-59):
       External calls:
       - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT.mul(multiplier)) (contracts/VestingMarketing.sol#47-50)
       - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT) (contracts/VestingMarketing.sol#53)
       Event emitted after the call(s):
       - Claimed(RELEASEAMOUNT,epoch) (contracts/VestingMarketing.sol#57)
Reentrancy in VestingTeam.claimTokens() (contracts/VestingTeam.sol#35-59):
        External calls:
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT.mul(multiplier)) (contracts/VestingTeam.sol#47-50)
        - kstToken.safeTransfer(beneficiary,RELEASEAMOUNT) (contracts/VestingTeam.sol#53)
        Event emitted after the call(s):
        - Claimed(RELEASEAMOUNT,epoch) (contracts/VestingTeam.sol#57)
Pragma version^0.8.6 (contracts/KST.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.6 is not recommended for deployment
VestingCommunityIncentives.setBeneficiary(address)._addy (contracts/VestingCommunityIncentives.sol#79) lacks a zero-check on :
              - beneficiary = _addy (contracts/VestingCommunityIncentives.sol#81)
VestingTeam.setBeneficiary(address)._addy (contracts/VestingTeam.sol#79) lacks a zero-check on :
              - beneficiary = _addy (contracts/VestingTeam.sol#81)
VestingTreasury.setBeneficiary(address)._addy (contracts/VestingTreasury.sol#79) lacks a zero-check on :
              - beneficiary = _addy (contracts/VestingTreasury.sol#81)
```

AUTOMATED TESTING

```
IDO.swapTokens() (contracts/IDO.sol#220-305) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= t1.swapStart && block.timestamp <= t1.swapEnd,IDO: t1 swap start/end not time) (contracts/IDO.sol#239-243)
        - require(bool,string)(block.timestamp >= t2.swapStart && block.timestamp <= t2.swapEnd,IDO: t2 swap start/end not time) (contracts/IDO.sol#249-253)
        - require(bool,string)(block.timestamp >= t3.swapStart && block.timestamp <= t3.swapEnd,IDO: t3 swap start/end not time) (contracts/IDO.sol#267-271)
        - require(bool,string)(block.timestamp >= t4.swapStart && block.timestamp <= t4.swapEnd,IDO: t4 swap start/end not time) (contracts/IDO.sol#277-281)
        - require(bool,string)(block.timestamp >= t5.swapStart && block.timestamp <= t5.swapEnd,IDO: t5 swap start/end not time) (contracts/IDO.sol#291-295)
IDO.claimTokens() (contracts/IDO.sol#307-315) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= claimStartTime,IDO: Claim start not yet.) (contracts/IDO.sol#310)
IDO.withdrawIDOToken() (contracts/IDO.sol#451-457) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= t5.swapEnd,IDO: Swap time not done yet) (contracts/IDO.sol#452)

VestingTreasury.claimTokens() (contracts/VestingTreasury.sol#35-59) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= startTime,start time not set) (contracts/VestingTreasury.sol#36)
        - require(bool,string)(block.timestamp >= lastClaimTime.add(CLAIM_DELAY),delay since last claim not passed) (contracts/VestingTreasury.sol#37-40)
        - block.timestamp > END_DAY (contracts/VestingTreasury.sol#41)
        - currentEpoch > epoch (contracts/VestingTreasury.sol#45)
VestingTreasury.getClaimAmount() (contracts/VestingTreasury.sol#61-69) uses timestamp for comparisons
        Dangerous comparisons:
        - currentEpoch > epoch (contracts/VestingTreasury.sol#63)
VestingTreasury.claimDust() (contracts/VestingTreasury.sol#71-77) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= END_DAY,Vesting not yet finished) (contracts/VestingTreasury.sol#72)

KSTStaking.lock(address,uint256) (contracts/KSTStaking.sol#59-65) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(userUnlockTime > block.timestamp,KSTStaking: unlock is in the past) (contracts/KSTStaking.sol#60)

VestingMarketing.claimTokens() (contracts/VestingMarketing.sol#35-59) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= startTime,start time not set) (contracts/VestingMarketing.sol#36)
        - require(bool,string)(block.timestamp >= lastClaimTime.add(CLAIM_DELAY),delay since last claim not passed) (contracts/VestingMarketing.sol#37-40)
        - block.timestamp > END_DAY (contracts/VestingMarketing.sol#41)
        - currentEpoch > epoch (contracts/VestingMarketing.sol#45)
VestingMarketing.getClaimAmount() (contracts/VestingMarketing.sol#61-69) uses timestamp for comparisons
        Dangerous comparisons:
        - currentEpoch > epoch (contracts/VestingMarketing.sol#63)
VestingMarketing.claimDust() (contracts/VestingMarketing.sol#71-77) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= END_DAY,Vesting not yet finished) (contracts/VestingMarketing.sol#72)

VestingTeam.claimTokens() (contracts/VestingTeam.sol#35-59) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= startTime,start time not set) (contracts/VestingTeam.sol#36)
        - require(bool,string)(block.timestamp >= lastClaimTime.add(CLAIM_DELAY),delay since last claim not passed) (contracts/VestingTeam.sol#37-40)
        - block.timestamp > END_DAY (contracts/VestingTeam.sol#41)
        - currentEpoch > epoch (contracts/VestingTeam.sol#45)
VestingTeam.getClaimAmount() (contracts/VestingTeam.sol#61-69) uses timestamp for comparisons
        Dangerous comparisons:
        - currentEpoch > epoch (contracts/VestingTeam.sol#63)
VestingTeam.claimDust() (contracts/VestingTeam.sol#71-77) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= END_DAY,Vesting not yet finished) (contracts/VestingTeam.sol#72)

KSMStarterLotteryRNG.returnLINK() (contracts/KSMStarterLotteryRNG.sol#55-57) ignores return value by LINK.transfer(msg.sender,LINK.balanceOf(address(this))) (contracts/KSMStarterLotteryRNG.sol#56)

IDO.setOperator(address) (contracts/IDO.sol#459-462) should emit an event for:
        - operator = _operator (contracts/IDO.sol#461)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control
INFO:Detectors:
IDO.setSwapPrice(uint256) (contracts/IDO.sol#440-444) should emit an event for:
        - swapPrice = _price (contracts/IDO.sol#443)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

KSMStarterLotteryRNG.requestRandomNumbers().requestId (contracts/KSMStarterLotteryRNG.sol#50) shadows:
        - KSMStarterLotteryRNG.requestId (contracts/KSMStarterLotteryRNG.sol#9) (state variable)

register() should be declared external:
        - IDO.register() (contracts/IDO.sol#182-218)
swapTokens() should be declared external:
        - IDO.swapTokens() (contracts/IDO.sol#220-305)
claimTokens() should be declared external:
        - IDO.claimTokens() (contracts/IDO.sol#307-315)
getTier1Registrants() should be declared external:
        - IDO.getTier1Registrants() (contracts/IDO.sol#403-405)
getTotalRaised() should be declared external:
        - IDO.getTotalRaised() (contracts/IDO.sol#417-422)
setOperator(address) should be declared external:
        - IDO.setOperator(address) (contracts/IDO.sol#459-462)
createIDO(uint256[15],address,address,address,uint256) should be declared external:
        - IDOFactory.createIDO(uint256[15],address,address,address,uint256) (contracts/IDOFactory.sol#11-26)
setOperator(address) should be declared external:
        - IDOFactory.setOperator(address) (contracts/IDOFactory.sol#28-32)
```

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:

KSMStarterLotteryRNG.sol, IDOFactory.sol, USDT.sol

Report for contracts/KSMStarterLotteryRNG.sol
https://dashboard.mythx.io/#/console/analyses/8817d39e-ad99-4309-bc1f-098a304c17e1

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.

Report for contracts/IDOFactory.sol
https://dashboard.mythx.io/#/console/analyses/3bae4e12-c5ab-4112-894a-e575e39d95d3

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
19	(SWC-123) Requirement Violation	Low	Requirement violation.

Report for contracts/USDT.sol
https://dashboard.mythx.io/#/console/analyses/2805a41e-89d5-418c-a507-582548967c76

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.

IDO.sol

Report for contracts/IDO.sol
https://dashboard.mythx.io/#/console/analyses/432f5699-6b9d-4e85-886a-87954a77e62f

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
82	(SWC-110) Assert Violation	Unknown	Out of bounds array access
84	(SWC-110) Assert Violation	Unknown	Out of bounds array access
88	(SWC-110) Assert Violation	Unknown	Out of bounds array access
93	(SWC-110) Assert Violation	Unknown	Out of bounds array access
97	(SWC-110) Assert Violation	Unknown	Out of bounds array access
99	(SWC-110) Assert Violation	Unknown	Out of bounds array access
101	(SWC-110) Assert Violation	Unknown	Out of bounds array access
103	(SWC-110) Assert Violation	Unknown	Out of bounds array access
104	(SWC-110) Assert Violation	Unknown	Out of bounds array access
110	(SWC-110) Assert Violation	Unknown	Out of bounds array access
111	(SWC-110) Assert Violation	Unknown	Out of bounds array access
112	(SWC-110) Assert Violation	Unknown	Out of bounds array access
113	(SWC-110) Assert Violation	Unknown	Out of bounds array access
114	(SWC-110) Assert Violation	Unknown	Out of bounds array access
116	(SWC-110) Assert Violation	Unknown	Out of bounds array access
117	(SWC-110) Assert Violation	Unknown	Out of bounds array access

KSTStaking.sol, TestERC20.sol

Report for contracts/KSTStaking.sol
https://dashboard.mythx.io/#/console/analyses/8d6c6aeb-1763-4536-9309-a5cfb02eb5cc

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
54	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered

Report for contracts/TestERC20.sol
https://dashboard.mythx.io/#/console/analyses/20bd812b-232c-4de3-af0a-30431f4f3aba

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.

KST.sol

Report for contracts/KST.sol
https://dashboard.mythx.io/#/console/analyses/b0d9f001-2010-445e-90e3-9d5698ad7f09

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.

All relevant valid findings were founded in the manual code review.

THANK YOU FOR CHOOSING

// HALBORN