

SMART CONTRACT AUDIT REPORT  
for  
**DSC MIX PROTOCOL**

TheGreatHB  
Oct 16th, 2021

# TABLE OF CONTENTS

---

1. Introduction
2. Summary
3. Finding
4. Test Results for All Files
5. Disclaimer
6. Conclusion

# 1. Introduction

---

이 보고서는 Doge Sound Club 팀(이하 DSC)이 작성한 Mix 프로토콜 스마트 계약의 보안을 감사하기 위해 작성되었습니다. 본 감사에서는 DSC 팀이 작성한 계약의 안정성과 보안성에 최우선을 두어 검토가 진행되었습니다. 또한 그 외에도, 백서의 내용이 잘 이행되었는지와 팀이 의도한 방향으로 설계가 잘 되었는지를 팀과의 지속적인 확인 과정을 거쳐서 스마트 계약의 감사 및 수정을 진행하였습니다. 수정된 부분들은 최종적으로 DSC 팀에 의해 확인 및 검토가 완료되었습니다.

## 1.1 About Mix

---

Mix 프로토콜은 2021 년 Mates NFT 를 출시한 DSC 에서 새롭게 출시하는 프로토콜입니다. Mix 토큰은 DSC 생태계에서 기본 화폐로 사용될 예정이며, DSC 의 NFT 인 Mates 와 OGs 의 이름을 변경하고, OGs 를 발행하고, DSC 와의 파트너십 조건으로 Mix 의 사용을 제안하며 DSC 의 최종 목표인 메타버스 생태계에서의 통화로 사용되는 등 여러 기능을 갖는 토큰입니다.

Mix 프로토콜의 기본 정보는 다음과 같습니다.

프로젝트명	Mix protocol
플랫폼	Klaytn
언어	Solidity
코드저장소	<a href="https://github.com/dogesoundclub/mix">https://github.com/dogesoundclub/mix</a>
오더 의뢰 커밋해시	ce4c28690ff855a79954284229af5771b262a4f8
백서	<a href="https://medium.com/dogesoundclub/dsc-mix-nft-%ED%97%88%EB%B8%8C%EB%A5%BC-%EC%9C%84%ED%95%9C-%ED%86%A0%ED%81%B0-3299dd3a8d1d">https://medium.com/dogesoundclub/dsc-mix-nft-%ED%97%88%EB%B8%8C%EB%A5%BC-%EC%9C%84%ED%95%9C-%ED%86%A0%ED%81%B0-3299dd3a8d1d</a>

# 1. Introduction

---

검토범위 :

- IBooth.sol
- IBurnPool.sol
- IForwardingPool.sol
- IKIP17Dividend.sol
- IKIP7StakingPool.sol
- IMix.sol
- IMixEmitter.sol
- ITurntableKIP17Listeners.sol
- ITurntableKIP7Listeners.sol
- ITurntables.sol
- IMixDividend.sol <삭제됨>
- Booth.sol
- BurnPool.sol
- ForwardingPool.sol
- DevFundToken.sol
- KIP17Dividend.sol
- KIP7StakingPool.sol
- Mix.sol
- MixEmitter.sol
- TurntableKIP17Listeners.sol
- TurntableKIP7Listeners.sol
- Turntables.sol
- MixDividend.sol <삭제됨>





# 1. Introduction

---

## 1.2 Vulnerability Severity Classification

---

본 감사에서는 스마트 계약의 문제점들을 하기의 4 종류로 분류했습니다.









 <b>Critical</b>	프로토콜이 멈출 수 있는 등 매우 큰 영향을 끼치는 문제
 <b>High</b>	Critical 보다는 약하지만 가볍게 넘길 수 없는 중대한 문제
 <b>Medium</b>	백서와 구현된 내용이 다르거나, 구현체가 개발자의 의도와는 다르게 잘못 구현된 정도의 문제
 <b>Low</b>	수정이 필수는 아니나, 해서 나쁠 것이 없는 정도의 문제

## 2. Summary

---

다음은 Mix 프로토콜의 설계 및 구현을 분석한 결과 요약입니다.

감사의 첫번째 단계에서는 스마트 계약 코드를 간단히 검토하며 일반적으로 알려진 코딩 버그를 찾습니다. 이후 설계의 무결성을 검토하고 DeFi 관련 측면의 문제점을 정밀 조사합니다. 테스트코드를 작성하며 가능한 많은 경우에서 문제없이 작동함을 테스트합니다.

문제의 심각성	문제의 개수	
 <b>Critical</b>	6	
 <b>High</b>	7	
 <b>Medium</b>	16	
 <b>Low</b>	4	
<b>총 합</b>	33	

자세한 내용은 다음 섹션(3. Finding) 에서 확인할 수 있습니다.

### 3. Finding

---

F0-1.

대상 : Turntables.sol, ITurntables.sol

심각성 :  **Critical**

커밋해시 : ce4c28690ff855a79954284229af5771b262a4f8

<코드 생략>

문제점 :

계약 내 백서에서 언급된 턴테이블의 배터리 관련 기능이 구현되어 있지 않습니다. 또한 Turntables 계약은 MixDividend 계약을 상속하는데, MixDividend 계약은 지갑 별로 보상을 계산합니다. Turntables 계약은 지갑 별 보상이 아닌 턴테이블의 id 별로 보상을 계산해야 하니 MixDividend 계약을 상속해서는 안됩니다.

해결방안 :

개발팀에 구현 및 수정을 요청했습니다.

수정 커밋해시 : 3072671eb98598f9ccca791b2ff57e203cf284bf

<수정 코드 생략>

### 3. Finding

---

F0-2.

대상 : Turntables.sol, ITurntables.sol

심각성 : ☒ **Medium**

커밋해시 : 3072671eb98598f9ccca791b2ff57e203cf284bf

<코드 생략>

문제점 :

계약 내 백서에서 언급된 턴테이블의 배터리 충전 기능이 구현되어 있지 않습니다.

해결방안 :

개발팀에 구현을 요청했습니다.

수정 커밋해시 : 9bafd325645e623d63b14bb6d3f19bc752824759

<수정 코드 생략>



## 3. Finding

---

F1.

대상 : KIP17Dividned.sol

심각성 :  **Medium**

커밋해시 : 75901df0f341e62ecea812ddd88445138d9bafa6

<코드>

```
function claim(uint256[] calldata ids) external {
    updateBalance();
    uint256 length = ids.length;
    uint256 totalClaimable = 0;
    for (uint256 i = 0; i < length; i = i.add(1)) {
        uint256 id = ids[i];
        require(nft.ownerOf(id) == msg.sender);
        uint256 claimable = _claimableOf(id);
        if (claimable > 0) {
            claimed[id] = claimed[id].add(claimable);
            emit Claim(id, claimable);
            mix.transfer(msg.sender, claimable);
            totalClaimable = totalClaimable.add(claimable);
        }
    }
    currentBalance = currentBalance.sub(totalClaimable);
}
```

문제점 :

계약 내 백서에서 언급된, 누적 보상을 수령하기 위해 보상의 10% 를 선납하는 과정이 구현되어 있지 않습니다.

해결방안 :

선납을 위해 prepayment 지역 변수를 추가하고 transferFrom 으로 선납금 차감 후 보상 지급하게 수정하였습니다. 선납금을 보상 수령의 자격 여부로 판단하여 같이 돌려주는 1 차 수정 후, 개발팀의 요청으로 선납금을 빼고 보상 금액만 돌려주는 2 차 수정을 진행하였습니다.

### 3. Finding

---

수정 커밋해시 :

1 차 수정 : cfe60a591c1647deb316a31f479cba7a3dbd6b86

```
function claim(uint256[] calldata ids) external returns (uint256 totalClaimable) {
    updateBalance();
    uint256 length = ids.length;
    for (uint256 i = 0; i < length; i = i.add(1)) {
        uint256 id = ids[i];
        require(nft.ownerOf(id) == msg.sender);
        uint256 claimable = _claimableOf(id);
        if (claimable > 0) {
            claimed[id] = claimed[id].add(claimable);
            emit Claim(id, claimable);
            totalClaimable = totalClaimable.add(claimable);
        }
    }
    uint256 prepayment = totalClaimable.div(10);
    mix.transferFrom(msg.sender, address(this), prepayment);
    mix.transfer(msg.sender, totalClaimable.add(prepayment));
    currentBalance = currentBalance.sub(totalClaimable);
}
```

2 차 수정 : ada62aadca06a2d202fae4a96fae29fddeb9fa5b

```
function claim(uint256[] calldata ids) external returns (uint256 totalClaimable) {

    <생략>

}
mix.burnFrom(msg.sender, totalClaimable.div(10));
mix.transfer(msg.sender, totalClaimable);
currentBalance = currentBalance.sub(totalClaimable);
}
```

### 3. Finding

---

F2.

대상 : TurntableKIP17Listeners.sol

심각성 :  **High**

커밋해시 : ce4c28690ff855a79954284229af5771b262a4f8

<코드>

```
function unlisten(uint256 turntableId, uint256[] calldata ids) external {
    updateBalance();
    uint256 length = ids.length;
    totalShares = totalShares.sub(length);
    for (uint256 i = 0; i < length; i = i.add(1)) {
        uint256 id = ids[i];
        require(nft.ownerOf(id) == msg.sender && listening[id] == true && listeningTo[id] ==
turntableId);
        shares[turntableId][id] = shares[turntableId][id].sub(1);
        pointsCorrection[turntableId][id] =
pointsCorrection[turntableId][id].add(int256(pointsPerShare));
        delete listeningTo[id];
        delete listening[id];
        emit Unlisten(turntableId, msg.sender, id);
    }
}
```

문제점 :

TurntableKIP17Listeners 에서 리스너가 보상을 요구할 때는 반드시 해당 nft 가 해당 턴테이블에 리스닝 중이어야 합니다. 하지만 unlisten 함수 호출 시 그동안의 누적 보상을 자동으로 수령하지 않고 리스닝이 끝나기 때문에, 이후 보상을 수령하려 해도 다시 해당 턴테이블에 리스닝 하지 않는 이상 수령이 불가능합니다.

해결방안 :

내부 \_claim 함수를 작성하고, unlisten 함수 호출 시 \_claim 함수를 호출하게 코드를 수정하였습니다.

### 3. Finding

---

수정 커밋해시 : 283cea80d8cfb861ee43f52b8c9e140e1f4d1cce

```
function unlisten(uint256 turntableId, uint256[] calldata ids) external {
    updateBalance();
    uint256 length = ids.length;
    totalShares = totalShares.sub(length);
    for (uint256 i = 0; i < length; i = i.add(1)) {
        uint256 id = ids[i];
        _claim(turntableId, id);
        shares[turntableId][id] = shares[turntableId][id].sub(1);
        pointsCorrection[turntableId][id] =
pointsCorrection[turntableId][id].add(int256(pointsPerShare));
        delete listeningTo[id];
        delete listening[id];
        emit Unlisten(turntableId, msg.sender, id);
    }
}
```

### 3. Finding

---

F3.

대상 : TurntableKIP17Listeners.sol, TurntableKIP7Listeners.sol

심각성 :  **High**

커밋해시 : 3747329e9698acf84fd4ff819d63119f6fd4170a

<코드>

```
function setTurntableFee(uint256 fee) onlyOwner external {  
    turntableFee = fee;  
}
```

문제점 :

turntableFee 변수에 넣을 수 있는 값의 제한이 없습니다. 10000 이 넘는 값이 입력된다면 단순히 보상을 못 받는 것이 아닌, 보상 계산 시 `_claim` 과 `claim` 함수 내의 `claimable.sub(fee)` 부분에서 `underflow` 가 발생하여 `claim` 관련 기능 등에서 프로토콜의 작동이 멈출 수 있습니다.

해결방안 :

각 계약의 `setTurntableFee` 함수에 입력된 `fee` 값이 10000 이상일 시 `revert` 가 발생하게 조건을 설정하였습니다.

(해당 문제는 단순 심각도만으로는 Critical 등급이나, ownership 소유자만 호출 가능한 함수이기에 High 등급으로 한단계 하향조정 하였습니다.)

수정 커밋해시 : 84a5e5dcbf7c937271dd9f133f4b4678503592d9

```
function setTurntableFee(uint256 fee) onlyOwner external {  
    require(fee < 1e4);  
    turntableFee = fee;  
}
```

### 3. Finding

---

F4.

대상 : TurntableKIP17Listeners.sol

심각성 :  Low

커밋해시 : 3747329e9698acf84fd4ff819d63119f6fd4170a

<코드>

```
function listen(uint256 turntableId, uint256[] calldata ids) external {
    updateBalance();
    uint256 length = ids.length;
    totalShares = totalShares.add(length);
    for (uint256 i = 0; i < length; i = i.add(1)) {
        uint256 id = ids[i];
        require(nft.ownerOf(id) == msg.sender);
        if (listening[id] == true) {
            uint256 originTo = listeningTo[id];
            _claim(originTo, id);
            shares[originTo][id] = shares[originTo][id].sub(1);
            pointsCorrection[originTo][id] = pointsCorrection[originTo][id].add(int256(pointsPerShare));
            emit Unlisten(originTo, msg.sender, id);
        }
        shares[turntableId][id] = shares[turntableId][id].add(1);
        pointsCorrection[turntableId][id] =
pointsCorrection[turntableId][id].sub(int256(pointsPerShare));
        listeningTo[id] = turntableId;
        listening[id] = true;
        emit Listen(turntableId, msg.sender, id);
    }
}
```

문제점 :

nft 가 특정 턴테이블에 리스닝 중이고 listen 함수를 호출하여 다시 재리스닝 시킬 경우에, 만약 이미 리스닝 중인 턴테이블에 재 리스닝을 시도한다면 불필요한 언리슨 - 리슨 반복이 일어납니다.

### 3. Finding

---

해결방안 :

재리스닝은 턴테이블이 변경되는 경우에만 일어나게 수정하였습니다.

수정 커밋해시 : a49bc57c7b580b51bc1291e29cc380953c18bd1b

```
function listen(uint256 turntableId, uint256[] calldata ids) external {  
  
    <생략>  
  
    require(nft.ownerOf(id) == msg.sender);  
    if (listening[id] == true && listeningTo[id] != turntableId) {  
        uint256 originTo = listeningTo[id];  
  
    <생략>  
  
    }  
}
```

### 3. Finding

---

F5.

대상 : Turntables.sol

심각성 :  **Medium**

커밋해시 : 3072671eb98598f9ccca791b2ff57e203cf284bf

<코드>

```
function claim(uint256[] memory turntableIds) public returns (uint256 totalClaimable) {
    updateBalance();

    uint256 toBurn = 0;
    uint256 length = turntableIds.length;
    for (uint256 i = 0; i < length; i = i.add(1)) {
        uint256 turntableId = turntableIds[i];
        require(ownerOf(turntableId) == msg.sender);
        uint256 claimable = _claimableOf(turntableId);
        if (claimable > 0) {
            uint256 realClaimable = 0;
            Turntable memory turntable = turntables[turntableId];
            if (turntable.endBlock <= turntable.lastClaimedBlock) {
                // ignore.
            } else if (turntable.endBlock < block.number) {
                realClaimable =
claimable.mul(turntable.endBlock.sub(turntable.lastClaimedBlock)).div(block.number.sub(turntable.las
tClaimedBlock));
            } else {
                realClaimable = claimable;
            }
            if (realClaimable > 0) {
                toBurn = toBurn.add(claimable.sub(realClaimable));
                claimed[turntableId] = claimed[turntableId].add(realClaimable);
                emit Claim(turntableId, realClaimable);
                totalClaimable = totalClaimable.add(realClaimable);

            }
            turntables[turntableId].lastClaimedBlock = block.number;
        }
    }
    mix.transfer(msg.sender, totalClaimable);
    mix.burn(toBurn);
    currentBalance = currentBalance.sub(totalClaimable.add(toBurn));
}
```



### 3. Finding

---

문제점 :

배터리가 모두 소모된 턴테이블에서 누적된 보상을 요청할 경우, 실제 받을 수 있는 보상은 보상 시점에 따라 달라집니다. 그 계산은 배터리가 소모되지 않았다는 가정하에 누적 보상을 계산하고, 마지막 보상 시점에 따라 계산된 보상의 일부 혹은 전량이 소각되고 남은 잔량을 받게 됩니다. 이때, 전량이 소각되는 경우에는 해당 값을 소각량에 더하지 않습니다. 따라서 그만큼은 소각이 일어나지 않고 `turntables` 계약 내에 토큰이 쌓이게 됩니다.

해결방안 :

전량이 소각되는 경우에도 총 소각량이 제대로 계산되게 코드를 수정하였습니다.

수정 커밋해시 : 6a90d8e440d2d73b8840a3213047c8787e28d35c

```
function claim(uint256[] memory turntableIds) public returns (uint256 totalClaimable) {  
  
    <생략>  
  
    realClaimable = claimable;  
    }  
    toBurn = toBurn.add(claimable.sub(realClaimable));  
    if (realClaimable > 0) {  
        claimed[turntableId] = claimed[turntableId].add(realClaimable);  
  
    <생략>  
  
}
```

### 3. Finding

---

F6.

대상 : Turntables.sol

심각성 :  **Medium**

커밋해시 : 9bafd325645e623d63b14bb6d3f19bc752824759

<코드>

```
function charge(uint256 turntableId, uint256 amount) external {
    require(amount > 0);
    Turntable storage turntable = turntables[turntableId];
    Type memory _type = types[turntable.typeId];
    uint256 chagedLifetime =
    _type.lifetime.mul(amount).mul(chargingEfficiency).div(100).div(_type.volume);
    uint256 oldEndBlock = turntable.endBlock;
    turntable.endBlock = (block.number < oldEndBlock ? oldEndBlock :
    block.number).add(chagedLifetime);

    mix.burnFrom(msg.sender, amount);
    emit Charge(msg.sender, turntableId, amount);
}
```

문제점 :

턴테이블의 배터리를 충전할 때, 본인의 턴테이블이 아닌 타인의 턴테이블을 충전할 수 있습니다. 이는 개발팀의 의도와는 다른 부분이며, 유저들의 실수를 야기할 가능성이 존재합니다.

해결방안 :

충전 시 자신의 턴테이블만 충전 가능하게 조건문을 추가하였습니다.

### 3. Finding

---

수정 커밋해시 : f593830a50a0e98211265cf4347ba32a28ac71af

```
function charge(uint256 turntableId, uint256 amount) external {  
    require(amount > 0);  
    Turntable storage turntable = turntables[turntableId];  
    require(turntable.owner == msg.sender);  
    Type memory _type = types[turntable.typeId];  
    <생략>  
}
```

### 3. Finding

---

F7.

대상 : Turntables.sol

심각성 :  **High**

커밋해시 : 9bafd325645e623d63b14bb6d3f19bc752824759

<코드>

```
function charge(uint256 turntableId, uint256 amount) external {
    require(amount > 0);
    Turntable storage turntable = turntables[turntableId];
    require(turntable.owner == msg.sender);
    Type memory _type = types[turntable.typeId];
    uint256 chagedLifetime =
_type.lifetime.mul(amount).mul(chargingEfficiency).div(100).div(_type.volume);
    uint256 oldEndBlock = turntable.endBlock;
    turntable.endBlock = (block.number < oldEndBlock ? oldEndBlock :
block.number).add(chagedLifetime);

    mix.burnFrom(msg.sender, amount);
    emit Charge(msg.sender, turntableId, amount);
}
```

문제점 :

턴테이블에서 배터리 충전 시, 계산식에 오류가 있습니다. 충전시간을 계산할 때 price 로 나누어 주어야 하는데 현재 volume 으로 나누어 주고 있습니다. 두 값의 차이는 약 10 의 18 승 이상 차이가 나므로, 1 초만 충전되어야 하는 경우에도 100000000000 년 이상이 충전됩니다.

해결방안 :

수식을 수정하였습니다.

### 3. Finding

---

수정 커밋해시 : 50e5f230b803973272ab7043dcd59eb38d46d3fc

```
function charge(uint256 turntableId, uint256 amount) external {  
  
    <생략>  
  
    uint256 chagedLifetime =  
    _type.lifetime.mul(amount).mul(chargingEfficiency).div(100).div(_type.price);  
  
    <생략>  
  
}
```

### 3. Finding

---

F8.

대상 : Turntables.sol

심각성 :  **High**

커밋해시 : 9bafd325645e623d63b14bb6d3f19bc752824759

<코드>

```
function charge(uint256 turntableId, uint256 amount) external {
    require(amount > 0);
    Turntable storage turntable = turntables[turntableId];
    require(turntable.owner == msg.sender);
    Type memory _type = types[turntable.typeId];
    uint256 chagedLifetime =
_type.lifetime.mul(amount).mul(chargingEfficiency).div(100).div(_type.price);
    uint256 oldEndBlock = turntable.endBlock;
    turntable.endBlock = (block.number < oldEndBlock ? oldEndBlock :
block.number).add(chagedLifetime);

    mix.burnFrom(msg.sender, amount);
    emit Charge(msg.sender, turntableId, amount);
}
```

문제점 :

턴테이블에서 배터리가 소진된 경우, claim 을 호출하지 않고 배터리를 충전하면 마치 그동안 배터리가 소진된 적이 없던 것처럼 누적된 보상을 다 받을 수 있는 문제가 있습니다. 해당 문제는 턴테이블의 lastClaimedBlock 값이 배터리가 소진되기 전의 시점에 멈춰 있기에 발생합니다. lastClaimedBlock 값이 변하지 않은 상태에서 충전이 되어 이후 보상을 수령할 시 마치 늘 배터리가 남아있던 것처럼 보상을 받을 수 있습니다.

해결방안 :

턴테이블을 충전할 때, 누적된 보상을 claim 하게 코드를 수정하였습니다. 배터리가 모두 소진된 턴테이블을 충전할 경우, 그간의 보상(일부)를 받거나 아무것도 받지 못하고, lastClaimedBlock 이 갱신되어 상기의 문제를 방지할 수 있습니다.

### 3. Finding

---

수정 커밋해시 : 54dc3fdcf64e74563dbc6e420dec6c2a3c6bcf27

```
function charge(uint256 turntableId, uint256 amount) external {  
  
    <생략>  
  
    Type memory _type = types[turntable.typeId];  
    uint256[] memory turntableIds = new uint256[](1);  
    turntableIds[0] = turntableId;  
    claim(turntableIds);  
    uint256 chargedLifetime =  
    _type.lifetime.mul(amount).mul(chargingEfficiency).div(100).div(_type.price);  
  
    <생략>  
  
}
```

### 3. Finding

---

F9.

대상 : IMixEmitter.sol

심각성 :  **Medium**

커밋해시 : 0d205a8c17f1bfd6da283c0d33e1ca4c1ce755fd

<코드>

```
function startBlock() external view returns (uint256);
```

문제점 :

IMixEmitter 계약을 상속하는 MixEmitter 계약에 startBlock view 함수가 존재하지 않기 때문에 컴파일 및 그에 따른 계약 배포가 불가능합니다.

해결방안 :

startBlock 함수는 started 함수로 변경되었습니다. IMixEmitter 계약의 startBlock 함수를 started 함수로 수정하였습니다.

수정 커밋해시 : dcbab7b9a1dff239b950dd63d437c1bce31b76e

```
function started() external view returns (bool);
```



### 3. Finding

---

F10.

대상 : IBurnPool.sol

심각성 :  **Medium**

커밋해시 : b2cf1e6b1dbfb46fb782b584b41e2dc983574d29

<코드>

```
function burn(uint256 amount) external;
```

문제점 :

IBurnPool 계약을 상속하는 BurnPool 계약에 burn 함수의 signature 가 동일하지 않습니다. 컴파일 및 그에 따른 계약 배포가 불가능합니다.

해결방안 :

burn 함수는 호출 시 일정량을 burn 하는 것이 아닌 계약 내 보유 중인 mix 토큰 전량을 burn 하는 함수로 변경됨에 따라 인자를 받지 않는 함수로 변경되었습니다. IBurnPool 계약의 burn(uint256 amount) 함수를 burn() 함수로 수정하였습니다.

수정 커밋해시 : cdf793bee3b07a92e7b171abd5228fe372af717d

```
function burn() external;
```

### 3. Finding

---

F11.

대상 : Turntables.sol, ITurntables.sol

심각성 :  **High**

커밋해시 : 129bf67197a961f25e546560f12f31d19ec2318b

<코드>

```
function ownerOf(uint256 turtableId) public returns (address) {  
    return turntables[turtableId].owner;  
}  
function exists(uint256 turtableId) external returns (bool) {  
    return turntables[turtableId].owner != address(0);  
}
```

문제점 :

Turntables 계약에서 ownerOf 함수와 exists 함수가 view 함수가 아닌 트랜잭션의 전송을 요구하는 일반 함수입니다. Client(App) 에서 해당 턴테이블의 소유주가 누구인지, 존재하는지를 확인할 수 없습니다.

해결방안 :

ownerOf 함수와 exists 함수를 view 로 수정하였습니다.

수정 커밋해시 : cedb862009a09b2a7354b26ac07227b09e658cc0

```
function ownerOf(uint256 turtableId) public view returns (address) {  
    return turntables[turtableId].owner;  
}  
function exists(uint256 turtableId) external view returns (bool) {  
    return turntables[turtableId].owner != address(0);  
}
```

## 3. Finding

---

F12.

대상 : Turntables.sol

심각성 :  **Critical**

커밋해시 : 129bf67197a961f25e546560f12f31d19ec2318b

<코드>

```
function buy(uint256 typeld) external returns (uint256 turntableId) {
    require(typeWhitelist[typeld]);
    Type memory _type = types[typeld];
    turntableId = turntables.length;
    turntables.push(
        Turntable({
            owner: msg.sender,
            typeld: typeld,
            endBlock: block.number.add(_type.lifetime),
            lastClaimedBlock: block.number
        })
    );
    updateBalance();
    totalVolume = totalVolume.add(_type.volume);
    pointsCorrection[turntableId] = int256(pointsPerShare.mul(_type.volume)).mul(-1);
    mix.transferFrom(msg.sender, address(this), _type.price);
    emit Buy(msg.sender, turntableId);
}

function charge(uint256 turntableId, uint256 amount) external {
    require(amount > 0);
    Turntable storage turntable = turntables[turntableId];
    require(turntable.owner == msg.sender);
    Type memory _type = types[turntable.typeld];
    uint256[] memory turntableIds = new uint256[](1);
    turntableIds[0] = turntableId;
    claim(turntableIds);
    uint256 chagedLifetime =
        _type.lifetime.mul(amount).mul(chargingEfficiency).div(100).div(_type.price);
    uint256 oldEndBlock = turntable.endBlock;
    turntable.endBlock = (block.number < oldEndBlock ? oldEndBlock :
        block.number).add(chagedLifetime);

    mix.burnFrom(msg.sender, amount);
    emit Charge(msg.sender, turntableId, amount);
}
```

### 3. Finding

---

```
function destroy(uint256 turntableId) external {
    Turntable memory turntable = turntables[turntableId];
    require(turntable.owner == msg.sender);
    uint256[] memory turntableIds = new uint256[](1);
    turntableIds[0] = turntableId;
    claim(turntableIds);
    Type memory _type = types[turntable.typeId];
    totalVolume = totalVolume.sub(_type.volume);
    delete pointsCorrection[turntableId];
    mix.transfer(msg.sender, _type.destroyReturn);
    mix.burn(_type.price - _type.destroyReturn);
    delete turntables[turntableId];
    emit Destroy(msg.sender, turntableId);
}
```

문제점 :

Turntables 계약에서 각각의 턴테이블들의 보상을 계산하는 방식은 MixEmitter 에서 보내온 토큰의 양을 체크하는 것이 아닌, 마지막 업데이트(MixEmitter 에서 토큰 전송을 요청하는 행위) 시의 토큰 양과 현재 업데이트 시의 토큰양을 비교하여 그 차액을 총 보상으로 계산하는 방식입니다. 하지만 현재 buy, charge, destroy 함수는 계약 내 mix 토큰의 양을 변화시키면서 계약에 기록된 토큰의 양(currentBalance)을 수정하지 않기에 정확한 보상량을 계산하는 것에 문제가 생깁니다. 이로 인해 향후 턴테이블의 토큰 보상 기능 및 destroy 기능이 아예 작동하지 않을 수 있습니다.

해결방안 :

buy, charge, destroy 함수의 마지막에 현재 계약 내 토큰 양을 currentBalance 변수에 업데이트해주는 것으로 해결했습니다. 또한 그로 인해 updateBalance 함수 내의 불필요한 currentBalance 변수 업데이트를 제거했습니다.

### 3. Finding

---

수정 커밋해시 : c944a92ee1e1834c6be64d0fbaf1b948b6f087ee

```
function buy(uint256 typeld) external returns (uint256 turntableId) {
```

<생략>

```
    mix.transferFrom(msg.sender, address(this), _type.price);
```

```
    currentBalance = mix.balanceOf(address(this));
```

```
    emit Buy(msg.sender, turntableId);
```

```
}
```

```
function charge(uint256 turntableId, uint256 amount) external {
```

<생략>

```
    mix.burnFrom(msg.sender, amount);
```

```
    currentBalance = mix.balanceOf(address(this));
```

```
    emit Charge(msg.sender, turntableId, amount);
```

```
}
```

```
function destroy(uint256 turntableId) external {
```

<생략>

```
    delete turntables[turntableId];
```

```
    currentBalance = mix.balanceOf(address(this));
```

```
    emit Destroy(msg.sender, turntableId);
```

```
}
```

### 3. Finding

---

F13.

대상 : Turntables.sol, ITurntables.sol

심각성 :  Low

커밋해시 : 129bf67197a961f25e546560f12f31d19ec2318b

<코드>

```
function turntableCount() external view returns (uint256) {  
    return turntables.length;  
}
```

문제점 :

turntableCount 함수의 이름이 모호합니다. 10 개의 턴테이블이 구매되었을 때, turntableCount 함수는 10 을 반환합니다. 하지만 1 개의 턴테이블이 분해되어 9 개가 남았을 때도 여전히 10 을 반환합니다.

해결방안 :

개발팀과의 협의 하에, 로직을 수정하는 것이 아닌 turntableLength 로 함수 이름을 수정하였습니다.

수정 커밋해시 : 1c34a8e213ee6d3c15a585a3411721df7d303f0d

```
function turntableLength() external view returns (uint256) {  
    return turntables.length;  
}
```

### 3. Finding

---

F14.

대상 : Turntables.sol

심각성 :  **Critical**

커밋해시 : 129bf67197a961f25e546560f12f31d19ec2318b

<코드>

```
constructor(IMixer _mixEmitter, uint256 _pid) public {
    mixEmitter = _mixEmitter;
    mix = _mixEmitter.mix();
    pid = _pid;
}
```

문제점 :

Turntables 계약이 배포되고 턴테이블들이 구매가 되었는데도 MixEmitter 의 Mix 토큰 분배가 시작되지 않았다면 보상 분배에 문제가 생깁니다. 보상 시작 블록 전에 구매한 턴테이블들은 lastClaimedBlock 이 그 이전으로 기록되어 이후 mix 보상 claim 시 정확한 계산을 위해 받아야 할 양보다 더 많은 양을 가져가게 됩니다. 그로 인해 언젠간 claim 시 underflow 가 발생하여 프로토콜이 멈출 수 있습니다.

해결방안 :

생성자에 조건문을 추가하여 MixEmitter 의 토큰 분배가 시작되었을 때만 배포가 가능하게 하였습니다.

수정 커밋해시 : b6d538d2349bf67dcc0f30271b42faf6bcec4765

```
constructor(IMixer _mixEmitter, uint256 _pid) public {
    require(_mixEmitter.started());
    mixEmitter = _mixEmitter;
    mix = _mixEmitter.mix();
    pid = _pid;
}
```

### 3. Finding

---

F15.

대상 : Turntables.sol

심각성 :  Low

커밋해시 : 129bf67197a961f25e546560f12f31d19ec2318b

<코드>

```
function claim(uint256[] memory turntableIds) public returns (uint256 totalClaimable) {
    updateBalance();

    uint256 toBurn = 0;
    uint256 length = turntableIds.length;

    <생략>

    mix.transfer(msg.sender, totalClaimable);
    mix.burn(toBurn);
    currentBalance = mix.balanceOf(address(this));
}
```

문제점 :

Turntables 계약에서 claim 함수 호출 시, 턴테이블들의 배터리 소진 여부에 따라 턴테이블 소유자에게 일정량의 mix 토큰을 보내주고, 나머지는 소각을 합니다. 이때 모든 턴테이블들의 배터리가 소진되지 않아서 소각할 양이 없더라도 0 개를 소각하고, 모든 턴테이블들의 배터리가 소진되어 보상으로 줄 양이 없더라도 0 개를 보내주는 불필요한 실행들이 있습니다.

해결방안 :

보상의 전송 및 소각에서 그 양이 0 보다 클 경우에만 실행되게 수정하였습니다.



### 3. Finding

---

수정 커밋해시 : 7f791fc5e02a101be2b322fbd906510db882c195

```
function claim(uint256[] memory turntableIds) public returns (uint256 totalClaimable) {
    updateBalance();

    uint256 toBurn = 0;
    uint256 length = turntableIds.length;

    <생략>


    if (totalClaimable > 0) mix.transfer(msg.sender, totalClaimable);
    if (toBurn > 0) mix.burn(toBurn);
    currentBalance = mix.balanceOf(address(this));
}
```

### 3. Finding

---

F16.

대상 : Turntables.sol

심각성 :  **High**

커밋해시 : 129bf67197a961f25e546560f12f31d19ec2318b

<코드>

```
function _accumulativeOf(uint256 turtableId) internal view returns (uint256) {  
    return uint256(int256(pointsPerShare).add(pointsCorrection[turtableId])).div(pointsMultiplier);  
}
```

문제점 :

Turntables 계약에서 턴테이블 소유주들이 보상을 받아갈 때, 그 보상은 턴테이블의 볼륨에 영향을 받아야 합니다. 하지만 현재 코드에서는 턴테이블의 볼륨을 고려하지 않기에 마치 모든 턴테이블의 볼륨이 1 인 것으로 가정하여 보상을 받아갑니다. 하지만 총 볼륨은 제대로 기록됩니다. 예를 들어 각각 볼륨이 100 인 A 턴테이블과 B 턴테이블만 존재한다고 했을 때, 둘은 각각 블록 당 보상의 100/200 씩 가져가야 하지만 둘 모두 1/200 씩 가져가고 나머지 198/200 은 Turntables 계약에 그대로 남게 됩니다.

해결방안 :

보상량을 계산할 때 해당 턴테이블의 볼륨을 고려하여 계산하게 계산식을 수정하였습니다.

수정 커밋해시 : 24533a9594fee28ee80f227737b390e24275b7fb

```
function _accumulativeOf(uint256 turtableId) internal view returns (uint256) {  
    return  
    uint256(int256(pointsPerShare.mul(types[turtables[turtableId]].typeId].volume)).add(pointsCorrection[turtableId])).div(pointsMultiplier);  
}
```

### 3. Finding

---

F17.

대상 : Turntables.sol

심각성 :  **Critical**

커밋해시 : 129bf67197a961f25e546560f12f31d19ec2318b

<코드>

```
function claim(uint256[] memory turntableIds) public returns (uint256 totalClaimable) {
    updateBalance();
    uint256 toBurn = 0;
    uint256 length = turntableIds.length;
    for (uint256 i = 0; i < length; i = i + 1) {
        uint256 turntableId = turntableIds[i];
        require(ownerOf(turntableId) == msg.sender);
        uint256 claimable = _claimableOf(turntableId);
        if (claimable > 0) {
            uint256 realClaimable = 0;
            Turntable memory turntable = turntables[turntableId];
            if (turntable.endBlock <= turntable.lastClaimedBlock) {
                // ignore.
            } else if (turntable.endBlock < block.number) {
                realClaimable = claimable.mul(turntable.endBlock.sub(turntable.lastClaimedBlock)).div(
                    block.number.sub(turntable.lastClaimedBlock)
                );
            } else {
                realClaimable = claimable;
            }

            toBurn = toBurn.add(claimable.sub(realClaimable));
            if (realClaimable > 0) {
                claimed[turntableId] = claimed[turntableId].add(realClaimable);
                emit Claim(turntableId, realClaimable);
                totalClaimable = totalClaimable.add(realClaimable);
            }
            turntables[turntableId].lastClaimedBlock = block.number;
        }
    }

    if (totalClaimable > 0) mix.transfer(msg.sender, totalClaimable);
    if (toBurn > 0) mix.burn(toBurn);
    currentBalance = mix.balanceOf(address(this));
}
```

### 3. Finding

---

문제점 :

배터리가 한번이라도 전부 소진된 턴테이블은 재충전 후 보상을 받을 시 무조건 더 많은 토큰을 가져가게 되어있습니다. 이는 `claimed` 변수와 보상 계산법에 의한 문제입니다.

```
claimed[turntableId] = claimed[turntableId].add(realClaimable)
```

배터리가 소진되어 보상이 적은 경우 `claimed` 변수에는 현재 소각량을 제외한 `realClaimable` 의 수치만 더해집니다.

원래는 100 토큰을 가져가야 하나 배터리 소진으로 인해 30 토큰을 가져가게 된 경우, 이후 재 충전을 진행하고 다시 100 토큰을 받아야 할 때, 받을 수 있는 `claimable` 값은 다음과 같이 계산됩니다.

```
function _claimableOf(uint256 turntableId) internal view returns (uint256) {  
    return _accumulativeOf(turntableId).sub(claimed[turntableId]);  
}
```

이때 `claimed` 는 사실 소각량을 제외하지 않은 값이어야 하기에  $200 - 100$  이 되어야 하지만, 현재는  $200 - 30$  으로 계산되어 더 많은 보상을 가져가게 됩니다. 이로 인해 `underflow` 가 발생하고 프로토콜이 작동을 멈출 수 있습니다.

해결방안 :

`realClaimed` 라는 변수를 추가하여, `realClaimed` 와 `claimed` 두가지 변수를 상황에 맞게 사용하는 것으로 수정하였습니다. 또한 `view` 함수인 `claimedOf` 도 `claimed` 가 아닌 `realClaimed` 값을 반환하는 것으로 하여, 향후 오해에 의해 잘 못 사용될 가능성을 막았습니다.

### 3. Finding

---

수정 커밋해시 : 8f22193a045c4cb2d698c9bd16af7f2623c5109c

```
mapping(uint256 => uint256) internal realClaimed;

function claimedOf(uint256 turntableId) public view returns (uint256) {
    return realClaimed[turntableId];
}

function claim(uint256[] memory turntableIds) public returns (uint256 totalClaimable) {

    <생략>

    toBurn = toBurn.add(claimable.sub(realClaimable));
    if (realClaimable > 0) {
        realClaimed[turntableId] = realClaimed[turntableId].add(realClaimable);
        emit Claim(turntableId, realClaimable);
        totalClaimable = totalClaimable.add(realClaimable);
    }

    claimed[turntableId] = claimed[turntableId].add(claimable);
    turntables[turntableId].lastClaimedBlock = block.number;
}

if (totalClaimable > 0) mix.transfer(msg.sender, totalClaimable);
if (toBurn > 0) mix.burn(toBurn);
currentBalance = mix.balanceOf(address(this));
}
```

### 3. Finding

---

F18.

대상 : Turntables.sol, ITurntables.sol

심각성 :  **Medium**

커밋해시 : 129bf67197a961f25e546560f12f31d19ec2318b

<코드>

```
uint256 internal totalVolume = 0;
```

문제점 :

Turntables 계약에서 totalVolume 변수는 현재 모든 턴테이블들의 볼륨 총 합을 나타내는 변수입니다. totalVolume 은 정확한 보상을 분배하기 위해 내부적으로 사용되며, Client(App) 에서도 경우에 따라 호출이 필요한 변수입니다. 하지만 현재 internal 변수로 선언이 되어 외부에서 접근이 불가능한 문제가 있습니다.

해결방안 :

해당 totalVolume 변수는 public 변수로 수정하였습니다.

수정 커밋해시 : f2433538dd82c45680fce4a0680c6ddffb922ecc

```
uint256 public totalVolume = 0;
```

### 3. Finding

---

F19.

대상 : KIP7StakingPool.sol

심각성 :  **Medium**

커밋해시 : 129bf67197a961f25e546560f12f31d19ec2318b

<코드>

```
function updateBalance() internal {
    if (totalShares > 0) {
        mixEmitter.updatePool(pid);
        uint256 balance = mix.balanceOf(address(this));
        uint256 value = balance.sub(currentBalance);
        if (value > 0) {
            pointsPerShare = pointsPerShare.add(value.mul(pointsMultiplier).div(totalShares));
            emit Distribute(msg.sender, value);
        }
        currentBalance = balance;
    }
}
```

문제점 :

MixEmitter 계약에서 보상 분배가 시작된 후, 최초의 유저가 KIP7StakingPool 계약에 최초로 staking 을 진행할 때 updateBalance 내부 함수가 호출됩니다. 이때 totalShares 가 0 이기에 MixEmitter 계약의 updatePool 함수를 실행하지 못해 그간의 mix 토큰 보상은 KIP7StakingPool 계약으로 넘어오지 않습니다. 이후 stake, unstake, claim 무엇이 되었든 updateBalance 함수를 호출하는 함수들이 호출된다면, 최초의 유저가 staking 을 한 시점 부터가 아닌, MixEmitter 계약에서 보상 분배가 시작된 이후부터 지금까지의 모든 보상이 한꺼번에 들어오게 되고, 그 보상을 최초의 유저 혼자 혹은 몇 명이서 독식하게 됩니다. 이 문제는 모든 유저가 unstaking 을 하고, 누군가 다시 첫번째 staking 을 진행할 때에도 발생합니다.

해결방안 :

최초의 유저가 staking 을 진행할 때, 그동안의 KIP7StakingPool 계약에 배정되었던 보상을 받아와서 소각을 진행하게 수정하였습니다. 이후 updateBalance 함수가 호출되었을 때, 최초의 staking 시점부터 호출 시점까지의 보상이 들어옵니다.

### 3. Finding

---

수정 커밋해시 : fdab526aa354adcab80c6e28b6bf6bbda7c5f6b9

```
function updateBalance() internal {
    if (totalShares > 0) {
        mixEmitter.updatePool(pid);
        uint256 balance = mix.balanceOf(address(this));
        uint256 value = balance.sub(currentBalance);
        if (value > 0) {
            pointsPerShare = pointsPerShare.add(value.mul(pointsMultiplier).div(totalShares));
            emit Distribute(msg.sender, value);
        }
        currentBalance = balance;
    } else {
        mixEmitter.updatePool(pid);
        uint256 balance = mix.balanceOf(address(this));
        uint256 value = balance.sub(currentBalance);
        if (value > 0) mix.burn(value);
    }
}
```



### 3. Finding

---

F20.

대상 : Turntables.sol

심각성 :  **Medium**

커밋해시 : 129bf67197a961f25e546560f12f31d19ec2318b

<코드>

```
function updateBalance() internal {
    if (totalVolume > 0) {
        mixEmitter.updatePool(pid);
        uint256 balance = mix.balanceOf(address(this));
        uint256 value = balance.sub(currentBalance);
        if (value > 0) {
            pointsPerShare = pointsPerShare.add(value.mul(pointsMultiplier).div(totalVolume));
            emit Distribute(msg.sender, value);
        }
    }
}
```

문제점 :

Turntables 계약에서 모든 턴테이블들이 분해된 경우에 발생할 수 있는 문제입니다. 모든 테이블이 분해되어 totalVolume 이 0 이 되었고, 어느정도 시간이 지난 이후에 한 유저가 Turntables 계약에서 턴테이블을 다시 새로이 구매했을 때 updateBalance 내부 함수가 호출됩니다. 이때 totalVolume 이 0 이기에 MixEmitter 계약의 updatePool 함수를 실행하지 못해 그간의 mix 토큰 보상은 Turntables 계약으로 넘어오지 않습니다. 이후 buy, charge, destroy, claim 무엇이 되었든 updateBalance 함수를 호출하는 함수들이 호출된다면, 새로이 턴테이블을 구매한 시점 부터가 아닌, 마지막 턴테이블이 분해된 시점부터의 모든 보상이 한꺼번에 들어오게 되고, 그 보상을 일부 유저가 독식하게 됩니다.

해결방안 :

totalVolume 이 0 일 때 updateBalance 함수가 호출된다면, 그동안의 Turntables 계약에 배정되었던 보상을 받아와서 소각을 진행하게 수정하였습니다.

### 3. Finding

---

수정 커밋해시 : c80cbf0282eb0b736e3e2c278162b5ebaa5cd3d5

```
function updateBalance() internal {
    if (totalVolume > 0) {
        mixEmitter.updatePool(pid);
        uint256 balance = mix.balanceOf(address(this));
        uint256 value = balance.sub(currentBalance);
        if (value > 0) {
            pointsPerShare = pointsPerShare.add(value.mul(pointsMultiplier).div(totalVolume));
            emit Distribute(msg.sender, value);
        }
    } else {
        mixEmitter.updatePool(pid);
        uint256 balance = mix.balanceOf(address(this));
        uint256 value = balance.sub(currentBalance);
        if (value > 0) mix.burn(value);
    }
}
```

### 3. Finding

---

F21.

대상 : TurntableKIP7Listeners.sol

심각성 :  **Medium**

커밋해시 : 129bf67197a961f25e546560f12f31d19ec2318b

<코드>

```
function updateBalance() private {
    if (totalShares > 0) {
        mixEmitter.updatePool(pid);
        uint256 balance = mix.balanceOf(address(this));
        uint256 value = balance.sub(currentBalance);
        if (value > 0) {
            pointsPerShare = pointsPerShare.add(value.mul(pointsMultiplier).div(totalShares));
            emit Distribute(msg.sender, value);
        }
        currentBalance = balance;
    }
}
```

문제점 :

MixEmitter 계약에서 보상 분배가 시작된 후, 최초의 유저가 TurntableKIP7Listeners 계약에 최초로 리스닝을 진행할 때 updateBalance 내부 함수가 호출됩니다. 이때 totalShares 가 0 이기에 MixEmitter 계약의 updatePool 함수를 실행하지 못해 그간의 mix 토큰 보상은 TurntableKIP7Listeners 계약으로 넘어오지 않습니다. 이후 listen, unlisten, claim 무엇이 되었든 updateBalance 함수를 호출하는 함수들이 호출된다면, 최초의 유저가 리스닝을 한 시점 부터가 아닌, MixEmitter 계약에서 보상 분배가 시작된 이후부터 지금까지의 모든 보상이 한꺼번에 들어오게 되고, 그 보상을 최초의 유저 혼자 혹은 몇 명이서 독식하게 됩니다. 이 문제는 모든 유저가 언리스닝을 하고, 누군가 다시 첫번째 리스닝을 진행할 때에도 발생합니다.

해결방안 :

최초의 유저가 리스닝을 진행할 때, 그동안의 TurntableKIP7Listeners 계약에 배정되었던 보상을 받아와서 소각을 진행하게 수정하였습니다. 이후 updateBalance 함수가 호출되었을 때, 최초의 리스닝 시점부터 호출 시점까지의 보상이 들어옵니다.

### 3. Finding

---

수정 커밋해시 : 3b997d36a916e87a8231b37479e580dad7a55fa5

```
function updateBalance() private {
  if (totalShares > 0) {
    mixEmitter.updatePool(pid);
    uint256 balance = mix.balanceOf(address(this));
    uint256 value = balance.sub(currentBalance);
    if (value > 0) {
      pointsPerShare = pointsPerShare.add(value.mul(pointsMultiplier).div(totalShares));
      emit Distribute(msg.sender, value);
    }
    currentBalance = balance;
  } else {
    mixEmitter.updatePool(pid);
    uint256 balance = mix.balanceOf(address(this));
    uint256 value = balance.sub(currentBalance);
    if (value > 0) mix.burn(value);
  }
}
```

## 3. Finding

---

F22.

대상 : TurntableKIP17Listeners.sol

심각성 :  **Medium**

커밋해시 : 129bf67197a961f25e546560f12f31d19ec2318b

<코드>

```
function updateBalance() private {
    if (totalShares > 0) {
        mixEmitter.updatePool(pid);
        uint256 balance = mix.balanceOf(address(this));
        uint256 value = balance.sub(currentBalance);
        if (value > 0) {
            pointsPerShare = pointsPerShare.add(value.mul(pointsMultiplier).div(totalShares));
            emit Distribute(msg.sender, value);
        }
        currentBalance = balance;
    }
}
```

문제점 :

MixEmitter 계약에서 보상 분배가 시작된 후, 최초의 유저가 TurntableKIP17Listeners 계약에 최초로 리스닝을 진행할 때 updateBalance 내부 함수가 호출됩니다. 이때 totalShares 가 0 이기에 MixEmitter 계약의 updatePool 함수를 실행하지 못해 그간의 mix 토큰 보상은 TurntableKIP17Listeners 계약으로 넘어오지 않습니다. 이후 listen, unlisten, claim 무엇이 되었든 updateBalance 함수를 호출하는 함수들이 호출된다면, 최초의 유저가 리스닝을 한 시점 부터가 아닌, MixEmitter 계약에서 보상 분배가 시작된 이후부터 지금까지의 모든 보상이 한꺼번에 들어오게 되고, 그 보상을 최초의 유저 혼자 혹은 몇 명이서 독식하게 됩니다. 이 문제는 모든 유저가 언리스닝을 하고, 누군가 다시 첫번째 리스닝을 진행할 때에도 발생합니다.

해결방안 :

최초의 유저가 리스닝을 진행할 때, 그동안의 TurntableKIP17Listeners 계약에 배정되었던 보상을 받아와서 소각을 진행하게 수정하였습니다. 이후 updateBalance 함수가 호출되었을 때, 최초의 리스닝 시점부터 호출 시점까지의 보상이 들어옵니다.

### 3. Finding

---

수정 커밋해시 : 3846e6545832dc237506c69503c62187bde4b72f

```
function updateBalance() private {
    if (totalShares > 0) {
        mixEmitter.updatePool(pid);
        uint256 balance = mix.balanceOf(address(this));
        uint256 value = balance.sub(currentBalance);
        if (value > 0) {
            pointsPerShare = pointsPerShare.add(value.mul(pointsMultiplier).div(totalShares));
            emit Distribute(msg.sender, value);
        }
        currentBalance = balance;
    } else {
        mixEmitter.updatePool(pid);
        uint256 balance = mix.balanceOf(address(this));
        uint256 value = balance.sub(currentBalance);
        if (value > 0) mix.burn(value);
    }
}
```

### 3. Finding

---

F23.

대상 : MixEmitter.sol, IMixEmitter.sol

심각성 :  Low

커밋해시 : 5b459df6b0e928c65db24d9ec8e39e93f9f5c140

<코드>

```
uint256 public emitPerBlock;

function setEmitPerBlock(uint256 _emitPerBlock) external onlyOwner {
    massUpdatePools();
    emitPerBlock = _emitPerBlock;
    emit SetEmitPerBlock(_emitPerBlock);
}
```

문제점 :

MixEmitter 계약에서 emitPerBlock 은 블록 당 mix 토큰의 발행량을 의미합니다. 이때 Emit 은 동사이기에 표현하고자 하는 의미를 파악하는데 어려움이 있을 수 있습니다.

해결방안 :

모호한 변수 및 함수 이름에서 오는 혼란을 방지하기 위해 emitPerBlock 변수, setEmitPerBlock 함수, SetEmitPerBlock 이벤트명의 emit 을 명사 형태인 emission 으로 수정하였습니다.

수정 커밋해시 : 53128843c331f730b78135a462c26bf4f73a431a

```
uint256 public emissionPerBlock;

function setEmissionPerBlock(uint256 _emissionPerBlock) external onlyOwner {
    massUpdatePools();
    emissionPerBlock = _emissionPerBlock;
    emit SetEmissionPerBlock(_emissionPerBlock);
}
```

### 3. Finding

---

F24.

대상 : TurntableKIP7Listeners.sol, ITurntableKIP7Listeners.sol

심각성 :  **Medium**

커밋해시 : 5b459df6b0e928c65db24d9ec8e39e93f9f5c140

<코드>

```
uint256 private totalShares = 0;  
uint256 private turntableFee = 300; // 1e4
```

문제점 :

TurntableKIP7Listeners 계약에서 totalShares 변수와 turntableFee 변수는 각각 현재 모든 리스너들의 amount 수치 총 합을 나타내는 변수와 그들이 받는 mix 보상의 수수료율(턴테이블의 소유주에게 가는 수수료)을 알려주는 변수입니다. 두 변수는 정확한 보상을 분배하기 위해 내부적으로 사용되며, Client(App) 에서도 경우에 따라 호출이 필요한 변수입니다. 하지만 현재 private 변수로 선언이 되어 외부에서 접근이 불가능한 문제가 있습니다.

해결방안 :

해당 totalShares 변수와 turntableFee 변수는 public 변수로 수정하였습니다.

수정 커밋해시 : a44889f16405e8da34ac6dacf4a1e4c5bf75e4e7

```
uint256 public totalShares = 0;  
uint256 public turntableFee = 300; // 1e4
```



### 3. Finding

---

F25.

대상 : TurntableKIP7Listeners.sol

심각성 :  **Medium**

커밋해시 : 5b459df6b0e928c65db24d9ec8e39e93f9f5c140

<코드>

```
function listen(uint256 turntableId, uint256 amount) external {
    updateBalance();
    totalShares = totalShares.add(amount);
    shares[turntableId][msg.sender] = shares[turntableId][msg.sender].add(amount);
    pointsCorrection[turntableId][msg.sender] = pointsCorrection[turntableId][msg.sender].sub(
        int256(pointsPerShare.mul(amount))
    );
    token.transferFrom(msg.sender, address(this), amount);
    emit Listen(turntableId, msg.sender, amount);
}
```

문제점 :

현재 TurntableKIP7Listeners 계약에서 존재하지 않는 턴테이블에게도 리스닝을 할 수 있습니다.

해결방안 :

존재하는 턴테이블에만 리스닝을 할 수 있게 조건문을 추가하였습니다.

수정 커밋해시 : 6b329aabb78709d5a03a3a1f5a60eb28c775167f

```
function listen(uint256 turntableId, uint256 amount) external {
    require(turntables.exists(turntableId));
    updateBalance();

    <생략>
}
```

### 3. Finding

---

F26.

대상 : TurntableKIP17Listeners.sol, ITurntableKIP17Listeners.sol

심각성 :  **Medium**

커밋해시 : 5b459df6b0e928c65db24d9ec8e39e93f9f5c140

<코드>

```
uint256 private totalShares = 0;  
uint256 private turntableFee = 300; // 1e4  
mapping(uint256 => uint256) private listeningTo;  
mapping(uint256 => bool) private listening;
```

문제점 :

TurntableKIP17Listeners 계약에서 totalShares, turntableFee, listeningTo, listening 변수는 각각 현재 모든 리스너들의 amount 수치 총 합, 보상의 수수료율, 현재 해당 nft가 리스닝 중인 턴테이블이 어느 것인지, 현재 리스닝 중인지를 알려주는 변수들입니다. 해당 변수들은 정확한 보상을 분배하기 위해, 정확한 수수료를 납부하기 위해 등등 계약에서 내부적으로 사용되며, Client(App) 에서도 경우에 따라 호출이 필요한 변수입니다. 하지만 현재 private 변수로 선언이 되어 외부에서 접근이 불가능한 문제가 있습니다.

해결방안 :

해당 totalShares, turntableFee, listeningTo, listening 변수들을 public 변수로 수정하였습니다.

수정 커밋해시 : 3b6d1be31a105e1d4b9a1c6cd7eb9ece139a38f6

```
uint256 public totalShares = 0;  
uint256 public turntableFee = 300; // 1e4  
mapping(uint256 => uint256) public listeningTo;  
mapping(uint256 => bool) public listening;
```

### 3. Finding

---

F27.

대상 : TurntableKIP17Listeners.sol

심각성 :  **High**

커밋해시 : 5b459df6b0e928c65db24d9ec8e39e93f9f5c140

<코드>

```
function listen(uint256 turntableId, uint256[] calldata ids) external {
    updateBalance();
    uint256 length = ids.length;
    totalShares = totalShares.add(length);
    for (uint256 i = 0; i < length; i = i + 1) {
        uint256 id = ids[i];
        require(nft.ownerOf(id) == msg.sender);
        if (listening[id] && listeningTo[id] != turntableId) {
            uint256 originTo = listeningTo[id];
            _claim(originTo, id);
            shares[originTo][id] = 0;
            pointsCorrection[originTo][id] = pointsCorrection[originTo][id].add(int256(pointsPerShare));
            emit Unlisten(originTo, msg.sender, id);
        }
        shares[turntableId][id] = 1;
        pointsCorrection[turntableId][id] =
pointsCorrection[turntableId][id].sub(int256(pointsPerShare));
        listeningTo[id] = turntableId;
        listening[id] = true;
        emit Listen(turntableId, msg.sender, id);
    }
}
```

문제점 :

TurntableKIP17Listeners 계약에서 이미 리스닝 중인 nft 에 listen 함수를 호출하여 다시 재리스닝 시킬 경우에, totalShare 가 늘어납니다. 리스닝 여부를 확인하는 조건절에 도착하기 전에 totalShares 가 증가되고, 재리스닝인 경우에는 다시 totalShares 를 감소시켜지 않습니다. 실제 수치와는 다르게 더 높아진 totalShares 는 모두에게 공평히 손해를 끼칩니다. 총 200 개의 nft 가 리스닝 중일 때, 각각의 nft 는 블록 당 TurntableKIP17Listeners 블록 보상의 1 / 200 씩 가져가야 하지만, 상기와 같은 경우가 반복된다면 1 / 300, 1 / 500 처럼 적게 가져가게 되어

### 3. Finding

---

모두 손실을 보고 잔여 Mix 는 누구도 가져갈 수 없게 계약에 쌓이게 됩니다.

해결방안 :

재리스닝인 경우에는 다시 totalShares 를 감소시키게 수정하였습니다.

수정 커밋해시 : 52b6e6c7100ca233b99a87ae780ca6d6109cf96e

```
function listen(uint256 turntableId, uint256[] calldata ids) external {  
  
    <생략>  
  
    if (listening[id] && listeningTo[id] != turntableId) {  
        uint256 originTo = listeningTo[id];  
        _claim(originTo, id);  
        shares[originTo][id] = 0;  
        totalShares = totalShares.sub(1);  
        pointsCorrection[originTo][id] = pointsCorrection[originTo][id].add(int256(pointsPerShare));  
        emit Unlisten(originTo, msg.sender, id);  
    }  
  
    <생략>  
  
}
```

### 3. Finding

---

F28.

대상 : TurntableKIP17Listeners.sol

심각성 :  **Critical**

커밋해시 : 5b459df6b0e928c65db24d9ec8e39e93f9f5c140

<코드>

```
function listen(uint256 turntableId, uint256[] calldata ids) external {
    updateBalance();
    uint256 length = ids.length;
    totalShares = totalShares.add(length);
    for (uint256 i = 0; i < length; i = i + 1) {
        uint256 id = ids[i];
        require(nft.ownerOf(id) == msg.sender);
        if (listening[id] && listeningTo[id] != turntableId) {
            uint256 originTo = listeningTo[id];
            _claim(originTo, id);
            shares[originTo][id] = 0;
            totalShares = totalShares.sub(1);
            pointsCorrection[originTo][id] = pointsCorrection[originTo][id].add(int256(pointsPerShare));
            emit Unlisten(originTo, msg.sender, id);
        }
        shares[turntableId][id] = 1;
        pointsCorrection[turntableId][id] =
pointsCorrection[turntableId][id].sub(int256(pointsPerShare));
        listeningTo[id] = turntableId;
        listening[id] = true;
        emit Listen(turntableId, msg.sender, id);
    }
}
```

문제점 :

TurntableKIP17Listeners 계약에서 특정 턴테이블에 리스닝 중인 nft 에 listen 함수를 호출하여 다시 재리스닝 시킬 경우에, 만일 동일 턴테이블에 재리스닝을 시켜준다면 이후 다시는 해당 nft 로 unlisten 과 claim 을 하지 못 하는 버그까지 일어납니다. 현재 재리스닝 여부를 확인하는 조건절은 같은 턴테이블이 아닐 경우만을 체크하고 있기에, 동일 턴테이블에 재리스닝을 할 경우 totalShare 가 늘어나도 pointsCorrection 이 감소하게 됩니다. 실제 수치와는 다르게 더 높아진 totalShares 는 모두에게 공평히 손해를 끼칩니다. 총 200 개의 nft 가 리스닝 중일

### 3. Finding

때, 각각의 nft 는 블록 당 TurntableKIP17Listeners 블록 보상의 1 / 200 씩 가져가야 하지만, 상기와 같은 경우가 반복된다면 1 / 300, 1/ 500 처럼 적게 가져가게 되어 모두 손실을 보고 잔여 Mix 는 누구도 가져갈 수 없게 계약에 쌓이게 됩니다. 또한 과도하게 감소한 pointsCorrection 은 이하의 공식으로 보상을 계산할 때, int256 타입의 값이 uint256 타입으로 변환할 수 없게 합니다. Unlisten 및 claim 함수가 호출될 때 mix 토큰 보상을 계산하는데 이때 underflow 가 발생할 수 있습니다. 만일 이미 pointsCorrection 이 지나치게 감소하여 underflow 가 발생해버렸다면, 유일한 해결책은 pointsPerShare 가 충분히 증가할 때까지 오랜 기간 기다리는 것뿐입니다. 물론 이 경우에도 영구적인 보상의 손실을 겪습니다.

```
uint256(int256(pointsPerShare.mul(shares[turtableId][id])).add(pointsCorrection[turtableId][id]))
```

해결방안 :

재리스닝의 경우에는 동일 테이블에 재리스닝을 할 수 없게 조건문을 추가하였습니다..

수정 커밋해시 : 6c85a23f4434d42d57ca63015d5c7426b94bcf80

```
function listen(uint256 turtableId, uint256[] calldata ids) external {

    <생략>

    if (listening[id] && listeningTo[id] != turtableId) {
        uint256 originTo = listeningTo[id];
        _claim(originTo, id);
        shares[originTo][id] = 0;
        totalShares = totalShares.sub(1);
        pointsCorrection[originTo][id] = pointsCorrection[originTo][id].add(int256(pointsPerShare));
        emit Unlisten(originTo, msg.sender, id);
    } else {
        require(!listening[id]);
    }
    shares[turtableId][id] = 1;

    <생략>
}
```

### 3. Finding

---

F29.

대상 : TurntableKIP17Listeners.sol

심각성 :  **Medium**

커밋해시 : 5b459df6b0e928c65db24d9ec8e39e93f9f5c140

<코드>

```
function listen(uint256 turntableId, uint256[] calldata ids) external {
    require(turntables.exists(turntableId));
    updateBalance();
    uint256 length = ids.length;
    totalShares = totalShares.add(length);
    for (uint256 i = 0; i < length; i = i + 1) {
        uint256 id = ids[i];
        require(nft.ownerOf(id) == msg.sender);
        if (listening[id] && listeningTo[id] != turntableId) {
            uint256 originTo = listeningTo[id];
            _claim(originTo, id);
            shares[originTo][id] = 0;
            totalShares = totalShares.sub(1);
            pointsCorrection[originTo][id] = pointsCorrection[originTo][id].add(int256(pointsPerShare));
            emit Unlisten(originTo, msg.sender, id);
        } else {
            require(!listening[id]);
        }
        shares[turntableId][id] = 1;
        pointsCorrection[turntableId][id] =
pointsCorrection[turntableId][id].sub(int256(pointsPerShare));
        listeningTo[id] = turntableId;
        listening[id] = true;
        emit Listen(turntableId, msg.sender, id);
    }
}
```

문제점 :

현재 TurntableKIP17Listeners 계약에서 존재하지 않는 턴테이블에게도 리스닝을 할 수 있습니다.

### 3. Finding

---

해결방안 :

존재하는 턴테이블에만 리스닝을 할 수 있게 조건문을 추가하였습니다.

수정 커밋해시 : f7f775de109648abaceb30ab4d5777cf4e772907

```
function listen(uint256 turntableId, uint256[] calldata ids) external {  
    require(turntables.exists(turntableId));  
    updateBalance();  
  
    <생략>  
}
```



### 3. Finding

---

F30.

대상 : TurntableKIP17Listeners.sol

심각성 :  **Critical**

커밋해시 : 41a9a56764fee4feef71d8237ecc7143fe87b781

<코드>

```
function _unlisten(uint256 turntableId, uint256 id) internal {
    uint256 lastIndex = listeners[turntableId].length.sub(1);
    uint256 index = listenersIndex[id];
    if (index != lastIndex) {
        uint256 last = listeners[turntableId][lastIndex];
        listeners[turntableId][index] = last;
        listenersIndex[last] = index;
    }
    listeners[turntableId].length--;

    _claim(turntableId, id);
    shares[turntableId][id] = false;
    pointsCorrection[turntableId][id] = pointsCorrection[turntableId][id].add(int256(pointsPerShare));
    emit Unlisten(turntableId, msg.sender, id);
}
```

문제점 :

TurntableKIP17Listeners 계약에서 리스너(nft)들 각각의 보상을 계산하는 방식은 MixEmitter 에서 보내온 토큰의 양을 체크하는 것이 아닌, 마지막 업데이트 (MixEmitter 에서 토큰 전송을 요청하는 행위) 시의 토큰 양과 현재 업데이트 시의 토큰양을 비교하여 그 차액을 총 보상으로 계산하는 방식입니다. 하지만 현재 unlisten 함수와 채리스닝 시의 listen 함수는 보상을 지급함으로써 계약 내 mix 토큰의 양을 변화시키지만 계약에 기록된 토큰의 양(currentBalance)은 수정하지 않기에 정확한 보상량을 계산하는 것에 문제가 생깁니다. 이로 인해 향후 리스너들의 토큰 보상 기능 및 unlisten 기능이 아예 작동하지 않을 수 있습니다.

### 3. Finding

---

해결방안 :

`_unlisten` 함수의 마지막에 현재 계약 내 토큰 양을 `currentBalance` 변수에 업데이트해주는 것으로 해결했습니다. 또한 `claim` 시 `updateBalance` 함수 내에서 `currentBalance` 변수 저장 방식을 연산 방식이 아닌 현재 수치를 받아오는 것으로 최적화 진행하였습니다.

수정 커밋해시 : `be5d8424fdf305faa1959a7dbe6d0145a72b8816`

```
function _unlisten(uint256 turntableId, uint256 id) internal {
    uint256 lastIndex = listeners[turntableId].length.sub(1);
    uint256 index = listenersIndex[id];
    if (index != lastIndex) {
        uint256 last = listeners[turntableId][lastIndex];
        listeners[turntableId][index] = last;
        listenersIndex[last] = index;
    }
    listeners[turntableId].length--;

    _claim(turntableId, id);
    shares[turntableId][id] = false;
    pointsCorrection[turntableId][id] = pointsCorrection[turntableId][id].add(int256(pointsPerShare));
    emit Unlisten(turntableId, msg.sender, id);
    currentBalance = mix.balanceOf(address(this));
}
```

### 3. Finding

---

F31.

대상 : TurntableKIP17Listeners.sol, ITurntableKIP17Listeners.sol

심각성 :  **Medium**

커밋해시 : 41a9a56764fee4feef71d8237ecc7143fe87b781

<코드>

```
function claimedOf(uint256 turntableId, uint256 id) public view returns (uint256) {  
    return claimed[turntableId][id];  
}
```

문제점 :

TurntableKIP17Listeners 계약에서 claimedOf 함수는 리스너(nft)가 이미 받아간 보상의 양을 확인하는데 사용되는 view 함수입니다. 하지만 현재의 claimedOf 함수는 턴테이블 소유자에게 가는 수수료를 고려하지 않았을 때의 수치이기에 실제 수령했던 보상량을 확인하려면 turntableFee 를 이용해 Client(App) 에서 추가 계산이 필요합니다. 또한 만약 프로토콜 중간에 turntableFee 가 변경된다면, 그때부터는 해당 시점을 따로 서버에 저장하는 등의 추가 작업을 하지 않으면 제대로 된 수령 보상량을 계산할 수 없습니다.

해결방안 :

실제 수령 보상량을 계산하기 위해 realClaimed 변수를 생성해주고, 보상 수령 시마다 해당 변수를 업데이트 하게 수정하였습니다. 또한 리스너의 claimed 값을 반환하던 claimed 함수를 실제 수령 보상량인 realClaimed 를 반환하는 realClaimed 함수로 수정하였습니다.

### 3. Finding

---

수정 커밋해시 : 9447e04419345e46af021a7dfacdd791fd916a4

```
mapping(uint256 => mapping(uint256 => uint256)) private realClaimed;

function realClaimedOf(uint256 turntableId, uint256 id) public view returns (uint256) {
    return realClaimed[turntableId][id];
}

function _claim(uint256 turntableId, uint256 id) internal returns (uint256 claimable) {
    require(nft.ownerOf(id) == msg.sender && listening[id] && listeningTo[id] == turntableId);
    claimable = _claimableOf(turntableId, id);
    if (claimable > 0) {
        claimed[turntableId][id] = claimed[turntableId][id].add(claimable);
        emit Claim(turntableId, id, claimable);
        uint256 fee = claimable.mul(turntableFee).div(1e4);
        if (turntables.exists(turntableId)) {
            mix.transfer(turntables.ownerOf(turntableId), fee);
        } else {
            mix.burn(fee);
        }
        mix.transfer(msg.sender, claimable.sub(fee));
        realClaimed[turntableId][id] = realClaimed[turntableId][id].add(claimable.sub(fee));
    }
}
```

## 4. Test Results for All Files

---

### Booth

- ✓ should be that stake function works well (477ms)
- ✓ should be that unstake function works well (344ms)

### BurnPool

- ✓ should be that burn function works well (525ms)

### ForwardingPool

- ✓ overall test (470ms)

### KIP17Dividend

- ✓ overall test (2132ms)

### KIP7StakingPool

- ✓ overall test (1612ms)

### Mix

- ✓ has given data (163ms)
- ✓ check the deployer balance (173ms)
- ✓ should be that only owner can set emitter and booth (276ms)
- ✓ should be that only emitter can mint mix token (209ms)
- ✓ should be that 0.3% of the amount of burning token goes to the booth (262ms)

## 4. Test Results for All Files

---

### MixEmitter

- ✓ should be that add / set functions work well (230ms)
- ✓ should be that when starting all pids' lastEmitBlock is changed to current block number (234ms)
- ✓ should be that pendingMix function returns value as correctly as possible (406ms)
- ✓ should be that update function works well (452ms)
- ✓ should be that setEmissionPerBlock function work properly (455ms)

### TurntableKIP17Listeners

- ✓ should be that setTurntableFee function works properly (855ms)
- ✓ should be that unwantedly minted mix token before the first listening should be burned (937ms)
- ✓ should be that if the turntable is not exist, listen is failed (1085ms)
- ✓ should be that only an owner of a nft token can listen / unlisten turntables with the nft (1017ms)
- ✓ should be that users can't claim with a nft token not listening to any turntables (910ms)
- ✓ should be that re-listening to the same turntable is reverted but to another turntable is not (946ms)
- ✓ should be that in cases of unlistening and re-listening to another turntable, rewards are claimed automatically (1092ms)
- ✓ should be that if a nft which is listening is transferred to another user, the new owner can claim accumulated rewards but the previous owner can't (1085ms)
- ✓ should be that even if the turntable is destroyed during listening, listeners can claim their mix and fee doesn't go to turntable's last owner but is burned (1132ms)
- ✓ should be that fee goes correctly when listeners claim their mix (1045ms)
- ✓ should be that fee goes to the turntable's owner even if its life is over (955ms)
- ✓ should be that if a turntable's owner is listening one's turntable, the owner can get all reward including fee (963ms)
- ✓ should be that listeners indexing works properly (1151ms)
- ✓ overall test (2621ms)

## 4. Test Results for All Files

---

### TurntableKIP7Listeners

- ✓ should be that setTurntableFee function works properly (729ms)
- ✓ should be that unwantedly minted mix token before the first listening should be burned (766ms)
- ✓ should be that if the turntable is not exist, listen is failed (946ms)
- ✓ should be that even if the turntable is destroyed during listening, listeners can claim their mix and fee is burned not goes to turntable's last owner (906ms)
- ✓ should be that fee goes correctly when listeners claim their mix (956ms)
- ✓ should be that fee goes to the turntable's owner even if its life is over (856ms)
- ✓ should be that if a turntable's owner is listening one's turntable, the owner can get all reward including fee (864ms)
- ✓ overall test (1358ms)

### Turntables

- ✓ should be that functions related with a type work properly (567ms)
- ✓ should be that buy, destroy, charge functions work properly (2945ms)
- ✓ should be that functions related with a claim work properly (1572ms)
- ✓ overall test (2265ms)

## 5. Disclaimer

---

이 보고서는 Doge Sound Club 개발팀이 작성한 Mix 프로토콜 스마트 계약의 보안을 감사하기 위해 작성되었습니다. 본 감사는 주어진 스마트 계약의 가능한 모든 보안 문제를 찾는 것에 대한 보증을 제공하지 않는다는 점에 유의해주시요. 즉, 평가 결과가 추가적인 보안 문제 혹은 알고리즘적 문제가 없음을 보장하지는 않습니다. 마지막으로 이 보안 감사가 투자 조언으로 사용되어서는 안됨을 알려드립니다.



## 6. Conclusion

---

본 감사에서는 자체적으로 구분한 Critical 단계의 문제점 6 개, High 단계의 문제점 7 개, Medium 단계의 문제점 16 개, Low 단계의 문제점 4 개를 찾았고, 그에 따른 수정을 진행하였습니다. 발견된 모든 문제점은 수정되었고, 수정된 코드에 대해서는 최종적으로 Doge Sound Club 개발팀에 의해 확인 및 검토가 완료되었습니다.