- **Risk 1: Users cannot make multiple claims**
- **Risk level: High**
- **Lines: PayoutRequest.sol,EscalationManager#119-125,124-135**

Question Description:

In this update, to address the issue of insufficient funds in a single risk pool, the project team has added a batch claim feature. However, due to a lack of rigorous code design, this could potentially lock users' funds. Specifically, within the initRequest function of the PayoutRequest contract, when a user calls the initRequest function to make a claim and the UMA oracle deems the claim valid, the optimisticOracle will invoke the assertionResolvedCallback function, setting policy.settled to true. If a user only makes a single claim, no issues will arise. However, if multiple claims are made against the same policyId, they will fail. This is because the line if (policy.settled) return; will cause the transaction to roll back, preventing subsequent claims from being processed successfully.

```
70  function initRequest(uint256 _policyId, uint256 _amount, address _to) public whenNotPaused returns (bytes32 assertionId) {
71      (address salesPolicy, , ) = ICapitalAgent(capitalAgent).getPolicyInfo();
72      ICapitalAgent(capitalAgent).updatePolicyStatus(_policyId);
73      uint256 _claimed = ICapitalAgent(capitalAgent).claimedAmount(salesPolicy, _policyId);
74      (uint256 _coverageAmount, , , bool _exist, bool _expired) = ISalesPolicy(salesPolicy).getPolicyData(_policyId);
75      require(_amount + _claimed <= _coverageAmount, "UnoRe: amount exceeds coverage amount");
76      require(_exist && !_expired, "UnoRe: policy expired or not exist");
77      Policy memory _policyData = policies[_policyId];
78      _policyData.insuranceAmount = _amount;
79      _policyData.payoutAddress = _to;
80      policies[_policyId] = _policyData;
81      if (!isUMAFailed) {
82          require(IERC721(salesPolicy).ownerOf(_policyId) == msg.sender, "UnoRe: not owner of policy id");
83          uint256 bond = optimisticOracle.getMinimumBond(address(defaultCurrency));
84          TransferHelper.safeTransferFrom(address(defaultCurrency), msg.sender, address(this), bond);
85          defaultCurrency.approve(address(optimisticOracle), bond);
86          assertionId = optimisticOracle.assertTruth(
87              abi.encodePacked(
88                  "Insurance contract is claiming that insurance event ",
89                  " had occurred as of ",
90                  ClaimData.toUtf8BytesUint(block.timestamp),
91                  "."
92              ),
93              _to,
94              address(this),
95              escalationManager,
96              uint64(assertionliveTime),
97              defaultCurrency,
98              bond,
99              defaultIdentifier,
100             bytes32(0) // No domain.
101         );
102         assertedPolicies[assertionId] = _policyId;
103         policiesAssertionId[_policyId] = assertionId;
104         emit InsurancePayoutRequested(_policyId, assertionId);
105     } else {
106         require(roleLockTime[msg.sender] <= block.timestamp, "RPayout: role lock time not passed");
107         require(msg.sender == claimsDao, "RPayout: can only called by claimsDao");
108         policies[_policyId].settled = true;
109         ssip.settlePayout(_policyId, _to, _amount);
110     }
111     isRequestInit[_policyId] = true;
112 }
```

```
113
114     function assertionResolvedCallback(bytes32 _assertionId, bool _assertedTruthfully) external whenNotPaused {
115         require(!isUMAFailed, "RPayout: pool failed");
116         require(msg.sender == address(optimisticOracle), "RPayout: !optimistic oracle");
117         // If the assertion was true, then the policy is settled.
118         uint256 _policyId = assertedPolicies[_assertionId];
119         if (_assertedTruthfully) {
120             // If already settled, do nothing. We don't revert because this function is called by the
121             // OptimisticOracleV3, which may block the assertion resolution.
122             Policy storage policy = policies[_policyId];
123             if (policy.settled) return;
124             policy.settled = true;
125             ssip.settlePayout(_policyId, policy.payoutAddress, policy.insuranceAmount);
126         } else {
127             isRequestInit[_policyId] = false;
128         }
129     }
130
```

In the EscalationManager contract, there is a lack of assignment and checking for the usage of the assertionId within the assertionResolvedCallback function.

```
123
124     function assertionResolvedCallback(bytes32 assertionId, bool assertedTruthfully) external override onlyRole(OPTMISTIC_ORACLE_V3_ROLE) {
125
126         AssertionApproval memory _assertionApproval = isAssertionIdApproved[assertionId];
127         if (_assertionApproval.exist) {
128             if (_assertionApproval.approved && !assertedTruthfully) {
129                 IPayoutRequest _payoutAddress = IPayoutRequest(OptimisticOracleV3Interface(msg.sender).getAssertion(assertionId).callbackRecipient);
130                 uint256 _policyId = _payoutAddress.assertedPolicies(assertionId);
131                 IPayoutRequest.Policy memory policy = _payoutAddress.policies(_policyId);
132                 ISingleSidedInsurancePool(_payoutAddress.ssip()).settlePayout(_policyId, policy.payoutAddress, policy.insuranceAmount);
133             } else if (assertedTruthfully && !_assertionApproval.approved) {
134                 revert("AssertionId not approved");
135             }
136         }
137     }
```

Modification suggestions:

1. It is recommended to remove the Policy check and policy.settled setting within the assertionResolvedCallback function, and also to inspect the usage of _assertionId to prevent its reuse.
2. In the EscalationManager contract, conduct a check on assertionId to prevent its reuse.