Smart Contract Audit

Date : 2024-11-23

System State Contract Audit

Executive Summary

Type	Smart Contract Audit
Audit Timeline	2 days
Runtime Environment	EVM
Languages	Solidity

Scope

./system_state_sc_Autovaults_V1_1

Summary of Findings

ID	Name	Description	Severity
H-01	Potential DOS and Scalability Issues Due to Unbounded Loops	The contract utilize loops that iterate over arrays (holders, usersWithDeposits) that can grow indefinitely	High
M-01	Inefficient Depositor Tracking	The isDepositor function in Autovaults V1.1 uses a linear search over the usersWithDeposits array to check if an address is a depositor	Medium

Findings

[H-01] Potential DOS and Scalability Issues Due to Unbounded Loops

Severity: High

Description:

The contract utilize loops that iterate over arrays (**holders**, **usersWithDeposits**) that can grow indefinitely. Functions like **distributeAutoVaultFee** and **updateParityAmount** can exceed gas limits as the arrays grow, making them uncallable. Attackers can use this to their advantage by creating a lot of addresses and depositing dust amounts in the protocol to grow the array

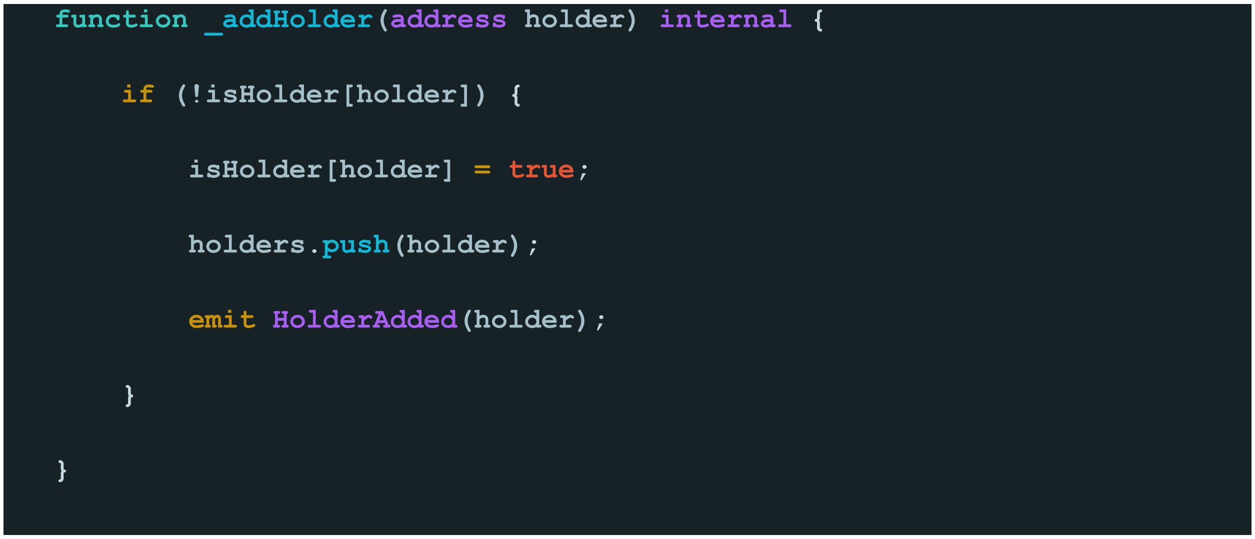
Impact:

- Critical functions may become uncallable due to gas limitations, leading to denial of service for essential contract operations and affecting the contract's functionality.

Recommendation:

1. Optimize Data Structures:
   a) Replace arrays with mappings to track holders, avoiding unbounded growth.
   b) Remove holders from tracking when their balance reaches zero.
2. Implement Pull Mechanism:
   a) Allow users to claim their rewards individually, reducing the need for loops.
   b) Store rewards in a mapping, and let users pull their rewards when needed.
3. Batch Processing:
   a) If looping is necessary, process in batches to stay within gas limits.

Proof of Concept



```
function _addHolder(address holder) internal {

    if (!isHolder[holder]) {

        isHolder[holder] = true;

        holders.push(holder);

        emit HolderAdded(holder);

    }

}
```

and

```
 function distributeAutoVaultFee(
       uint256 AutoVaultFee,
       address excludeUser
   ) private {
       uint256 totalSupply = DAVPLS.totalSupply();
       uint256 excludeUserBalance = DAVPLS.balanceOf(excludeUser);

       if (totalSupply == 0 || AutoVaultFee == 0) {
           return;
       }

       uint256 totalDistributableSupply =
totalSupply.sub(excludeUserBalance);

       uint256 holdersLength = DAVPLS.holdersLength();

       for (uint256 i = 0; i < holdersLength; i++) {
           address user = DAVPLS.holders(i);
           uint256 userBalance = DAVPLS.balanceOf(user);

           if (user == excludeUser) {
               continue;
           }

           if (userBalance > 0 && totalDistributableSupply > 0) {
               uint256 userShare = AutoVaultFee.mul(userBalance).div(
                   totalDistributableSupply
               );
               userAutoVault[user] = userAutoVault[user].add(userShare);

               emit AutoVaultFeeDistributed(user, userShare);
           }
       }
   }
```

[M-01] Inefficient Depositor Tracking

Severity: Medium

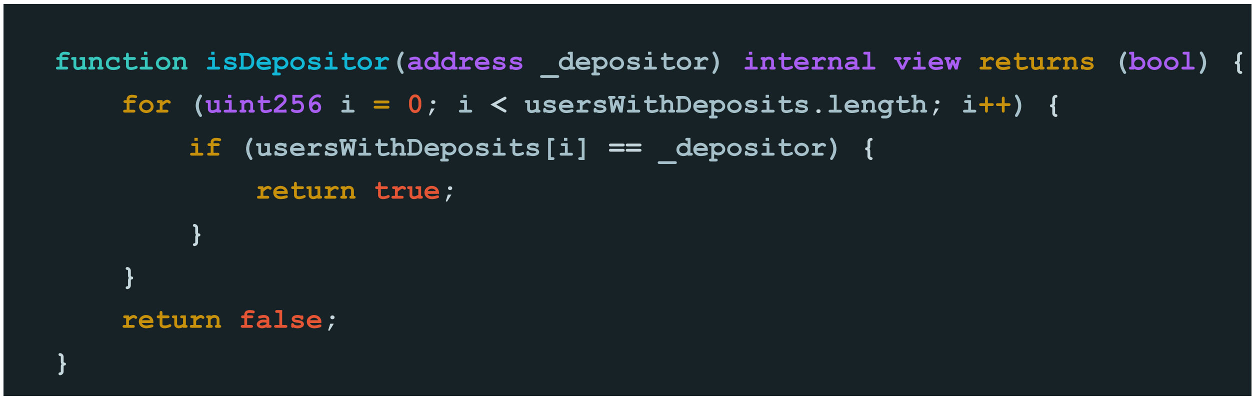
Description:

The **isDepositor** function in **Autovaults V1.1** uses a linear search over the **usersWithDeposits** array to check if an address is a depositor, which is inefficient for large arrays.

Impact:

As the number of depositors increases, the gas cost for this function will rise, leading to higher transaction fees and potential performance issues.

Proof of Concept :



```
function isDepositor(address _depositor) internal view returns (bool) {
    for (uint256 i = 0; i < usersWithDeposits.length; i++) {
        if (usersWithDeposits[i] == _depositor) {
            return true;
        }
    }
    return false;
}
```

Recommendation

Use a mapping for constant-time lookup of depositors.

Implementing a mapping:

```
mapping(address => bool) private isDepositorMapping;

function deposit(uint256 value) public onlyDepositer {
   // ...
   if (!isDepositorMapping[msg.sender]) {
       isDepositorMapping[msg.sender] = true;
       usersWithDeposits.push(msg.sender);
       NumberOfUser++;
   }
   // ...
}

function isDepositor(address _depositor) internal view returns (bool) {
   return isDepositorMapping[_depositor];
}
```