Smart Contract Audit

Date : 2024-11-23

Dav Token Audit

Executive Summary

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

Scope

./DAVTOKEN

Summary of Findings

ID	Name	Description	Severity
H-01	Potential DOS and Scalability Issues Due to Unbounded Loops	Both contracts utilize loops that iterate over arrays (holders, usersWithDeposits) that can grow indefinitely	High
M–01	Token Price Constants Are Hardcoded	The DAVTOKEN contract uses hardcoded price constants. This limits flexibility in updating token prices without deploying a new contract.	Medium
M-02	Missing Withdrawal Functionality in DAVTOKEN	The DAVTOKEN contract does not include a function to withdraw mistakenly sent tokens or Ether. This could result in the contract holding unintended balances	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] Token Price Constants Are Hardcoded

Severity: Medium

Description:

The **DAVTOKEN** contract uses hardcoded price constants (e.g., PFENIX_PRICE_ONE_TOKEN, PRICE_TWO_TOKEN). This limits flexibility in updating token prices without deploying a new contract.

Impact:

Any future adjustments to token prices will require redeployment, increasing operational complexity and cost.

Recommendation

Allow prices to be updated via a set-price function, restricted to the contract owner

```
uint256 public priceTwoToken;

function setPrice(uint256 _priceTwoToken) external onlyOwner {
   priceTwoToken = _priceTwoToken;
}
```

Proof Of Concept :

```
   uint256 public constant PRICE_TWO_TOKEN = 500000 ether;
```

[M-02] Missing Withdrawal Functionality in DAVTOKEN

Severity: Medium

Description:

The **DAVTOKEN** contract does not include a function to withdraw mistakenly sent tokens or Ether. This could result in the contract holding unintended balances.

Impact:

If tokens or Ether are mistakenly sent to the contract, they could become permanently locked.

Recommendation

Implement a withdrawal function to allow the owner to recover Ether or tokens: