Logo

Smart Contract Vulnerability Report

Vulnerability Title

Out of Gas error with Update Whitelist & Lack of Input Validation

---

Report Type

Smart Contract

Target

GitHub

Asset

SmartContractName.sol

Rating

Severity: Critical **Impact: Low I**

Description

For updating the whitelist, due to the lack of input size restrictions, the function cannot handle more than **840** addresses. If **841** or more addresses are provided, it results in an **out-of-gas error** and the transaction reverts. This causes the function to stop working (locked). This issue is caused by the absence of proper limits on the input array size.

The function responsible for adding or removing addresses from the whitelist lacks proper validation checks. It does not prevent invalid addresses, such as the zero address (address(0)), from being added or removed. Additionally, there is no mechanism to detect or prevent duplicate addresses.

Impact

- This bug can effectively disable the function by passing a large input array, leading to a complete `lock` of its execution. As a result, the protocol's security mechanism that relies on this function (such as whitelist control or pause/unpause capability) becomes entirely non-functional.

Vulnerability Details

```
//File:  ynBase.sol
//Line:118
//Line:141

    function _updatePauseWhitelist(address[] memory whitelistedForTransfers, bool
whitelisted) internal {
        ynBaseStorage storage $ = _getYnBaseStorage();
        for (uint256 i = 0; i < whitelistedForTransfers.length; i++) {
            address targetAddress = whitelistedForTransfers[i];
            $.pauseWhiteList[targetAddress] = whitelisted;
            emit PauseWhitelistUpdated(targetAddress, whitelisted);
        }
    }
```

🧪 Proof of Concept (PoC)

foundry test

```


//❌out of Gas revert
//❌Invalid Address & Duplicate Addresses



/*
This is a mock implementation of this functions from the ynBase.sol :
addToPauseWhitelist,
removeFromPauseWhitelist,
_updatePauseWhitelist
*/
pragma solidity ^0.8.24;

import "forge-std/Test.sol";
import "forge-std/console.sol";



// This mock simulates the three functions from the ynBase file
// for easier testing — corresponding to lines 118 to 141.
contract ynBase_mock{
    event TransfersUnpaused();
    event PauseWhitelistUpdated(address indexed addr, bool whitelisted);

    struct ynBaseStorage {
        mapping (address => bool) pauseWhiteList;
        bool transfersPaused;
    }
    ynBaseStorage private store;

//Here, we did not simulate the access control level because the issue is not with
access control,
// but with the function input structure
    modifier onlyRole() {
        _;
    }

    bytes32 private constant ynBaseStorageLocation =
0x7e7ba5b20f89141f0255e9704ce6ce6e55f5f28e4fc0d626fc76bedba3053200;

    function _getYnBaseStorage() private pure returns (ynBaseStorage storage $) {
        assembly {
            $.slot := ynBaseStorageLocation
        }
    }

// adding this  function for read Whitelisted
    function isWhitelisted(address addr) external view returns (bool) {
        ynBaseStorage storage $ = _getYnBaseStorage();
        return $.pauseWhiteList[addr];
    }

// Here, we did not simulate the access control level because the issue is not
with access control,
// but with the function input structure — specifically, the lack of restrictions
on address inputs.
    function addToPauseWhitelist(address[] memory whitelistedForTransfers)
external onlyRole {
        _updatePauseWhitelist(whitelistedForTransfers, true);
    }

    function removeFromPauseWhitelist(address[] memory unlisted) external onlyRole
{
        _updatePauseWhitelist(unlisted, false);
    }

    function _updatePauseWhitelist(address[] memory whitelistedForTransfers, bool
whitelisted) internal {
        ynBaseStorage storage $ = _getYnBaseStorage();
        for (uint256 i = 0; i < whitelistedForTransfers.length; i++) {
            address targetAddress = whitelistedForTransfers[i];
            $.pauseWhiteList[targetAddress] = whitelisted;
            emit PauseWhitelistUpdated(targetAddress, whitelisted);
        }
    }


}
```

```
contract Test_ynBase is  Test {
    ynBase_mock ynBase ;
    address[] public input;

    function setUp() public {
        ynBase = new ynBase_mock();

    }



// ⚠ When more than ~840 addresses are provided as input, the function reverts
due to out-of-gas,

// indicating a design flaw that leads to a (DoS) via excessive gas consumption.
// Since the loop performs expensive operations (SSTORE + event emission) per
address,
// large input arrays can exceed the block gas limit.
// As a result, the whitelist/blacklist cannot be updated, effectively locking the
system — a DoS condition.
//⚠ This is caused by missing input validation, which allows unbounded input and
leads to out-of-gas reverts.

    function test_GasOptimization() public {
        // ⚠Up to 840 addresses can be added to the list successfully,
        // ⚠but if more than that (e.g., 841 addresses) are provided,
         // a (DoS) occurs and the function reverts with an out-of-gas error.

        for (uint256 i = 0; i < 840; i++) {
            input.push(address(uint160(i + 1)));

            uint256 gasBefore = gasleft();
            ynBase.addToPauseWhitelist(input);
            uint256 gasAfter = gasleft();

//"Gas usage will increase over time due to the lack of input size limitations."
            uint256 gasUsed = gasBefore - gasAfter;
            console.log("Gas used for adding address", i, ":", gasUsed);
        }
//"This is to check whether an address has been added to the whitelist or not."
        for (uint256  i = 0; i < input.length; i++) {
            address target = input[i];
            bool isInWhitelist = ynBase.isWhitelisted(target);
            console.log("isWhitelisted:", isInWhitelist);
        }

    }

/*
Is restricting a function to users with a specific role sufficient? No — this is a
misconception.
Even if access is limited to certain roles, input validation is still necessary.

So far, based on the current mechanism, it is possible to add invalid addresses —
or even the same address multiple times — to the list without any input
validation.
*/

// ⚠ Invalid or zero addresses can be added to the list
// due to the lack of proper input validation.

    function test_AddInvalidAddress () public {
        input.push(address(0x0000000));
        ynBase.addToPauseWhitelist(input);
        bool isInWhitelist = ynBase.isWhitelisted(address(0x0000000));
        console.log("Address at index 0 isWhitelisted:", isInWhitelist);
    }


// ⚠ Duplicate or identical addresses can be added multiple times
// due to the lack of proper input validation.

    function test_AddingSameAddress() public {
        input.push(address(0x01));
        input.push(address(0x01));
        input.push(address(0x01));

        ynBase.addToPauseWhitelist(input);

        for (uint256 i = 0; i < input.length; i++) {
            address target = input[i];
            bool isInWhitelist = ynBase.isWhitelisted(target);
            console.log("isWhitelisted:", isInWhitelist);
        }
    }



}
```

How to fix it (Recommended)

```
// you can use this function for that
function _updatePauseWhitelist(address[] memory whitelistedForTransfers, bool
whitelisted) internal {
    require(whitelistedForTransfers.length <= 840, "Too many addresses");
    ynBaseStorage storage $ = _getYnBaseStorage();
    for (uint256 i = 0; i < whitelistedForTransfers.length; i++) {
        address targetAddress = whitelistedForTransfers[i];
        if (targetAddress == address(0)) {
            revert("Invalid address: zero address is not allowed");
        }
        if ($.pauseWhiteList[targetAddress] == whitelisted) {
            continue;
        }
        $.pauseWhiteList[targetAddress] = whitelisted;
        emit PauseWhitelistUpdated(targetAddress, whitelisted);
    }
}
```

🔗 References

🔗 vscode.blockscan 🔗 etherscan