Code Security Audit Report

For

SubStanceX

Nov 9th 2023

Damocles

Table of Contents

Summary

This report has been prepared for SubStanceX to discover issues and vulnerabilities in the source code of the SubStanceX project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following consideration:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Damocles

Overview

Audit Scope

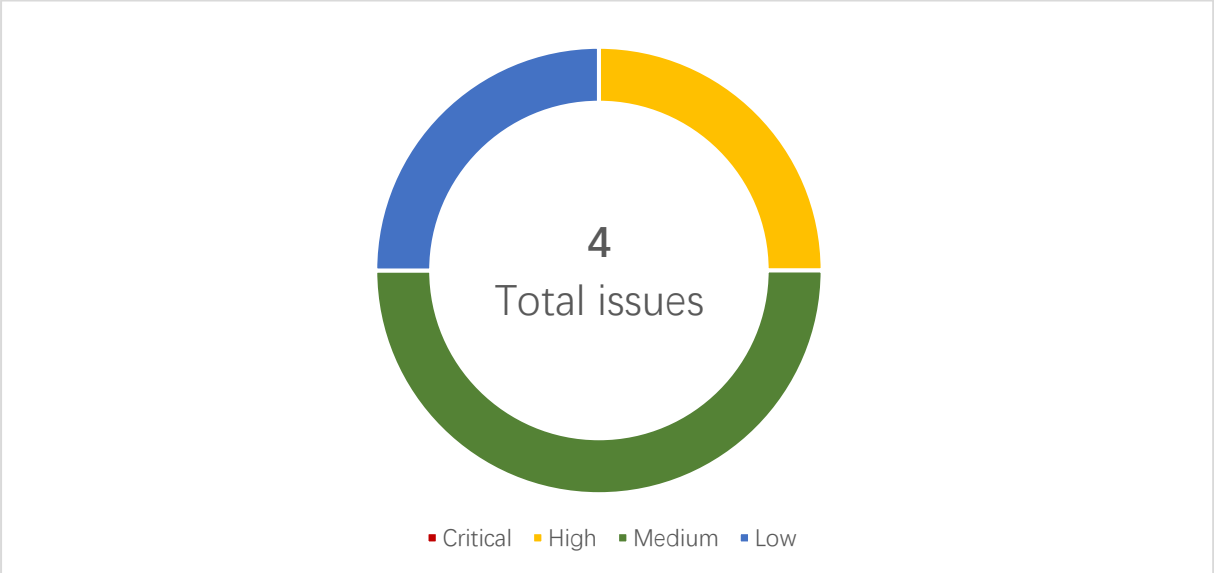
Contract Name	ChiliSwapContract
Platform	Etherum
Language	Solidity
Code Base	https://github.com/ElijahYao/SubstanceExchangeV1/tree/feature/v5_test
Commit	141540ca5c89b001df5ae43515b6515d9b896482

Result Summary

Vulnerability Level	Total	Pending	Solved	Acknowledged
Critical	0	0	0	0
High	1	0	1	0
Medium	2	0	2	0
Low	1	0	1	0

Damocles

Audit Result



SSX-01(High): x.Delegatehub _aggregate DOS Vulnerability

Category	Severity	Location	Status
Code Issue	High	DelegationHub.sol:125	**Solved**

Description

The DelegateHub _aggregate method performs a validation at the end to check if msg.value + receivedEth

is equal to valAccumulator. If they are not equal, the method will revert. However, there is a potential security vulnerability due to the receivedEth variable being a global variable that can be modified by the UB contract. By utilizing the UB contract to transfer a certain

amount of ETH, it is possible to cause a scenario where msg.value + receivedEth

and valAccumulator are never equal.

Vulnerability Analysis

1. In the aggregate function, after the delegate call is completed, there is a validation check if (msg.value + receivedEth != valAccumulator) and if it evaluates to true, the function will revert. Here, msg.value represents the value sent by the user during their function call, and valAccumulator is the cumulative sum of the values sent by the user. However, receivedEth is derived from the ETH sent to the DelegateHub contract from the UB contract.

```
111
112     function _aggregate(address sender, Call[] calldata calls) internal returns (Result[] memory returnData) {
113         if (senderOverride != address(0)) {
114             revert DelegationHub__ReentrantCall();
115         }
116         senderOverride = sender;
117         uint256 valAccumulator;
118         uint256 length = calls.length;
119         returnData = new Result[](length);
120         Call calldata calli;
121         for (uint256 i; i < length; ) {
122             Result memory result = returnData[i];
123             calli = calls[i];
124             uint256 val = calli.value;
125             unchecked {
126                 valAccumulator += val;
127             }
128             (bool success, bytes memory retData) = calli.target.call{value: val}(calli.payload);
129             if (!success && !calli.allowFailure) {
130                 if (retData.length == 0) revert();
131                 assembly {
132                     revert(add(0x20, retData), mload(retData))
133                 }
134             } else {
135                 result.success = success;
136                 result.returnData = retData;
137             }
138             unchecked {
139                 ++i;
140             }
141         }
142         if ((msg.value + receivedEth) != valAccumulator) {
143             revert DelegationHub__ValueMismatch();
144         }
145         receivedEth = 0;
146         senderOverride = address(0);
147     }
148
```

2. In the UB contract, there is a function that allows modification of the global variable receivedEth. This poses a significant security risk as it can lead to a situation where the validation check in the

DelegateHub contract fails, resulting in a revert of the transaction.

Recommendation

Using a temporary variable to store the value of ETH transferred from the UB contract to the Hub contract during user transactions is indeed a recommended approach for addressing the vulnerability. By doing so, you can compare the stored value with msg.value + receivedEth to check for any discrepancies.

SSX-02(Medium):x.Delegatehub SetDelegate Binding Issue

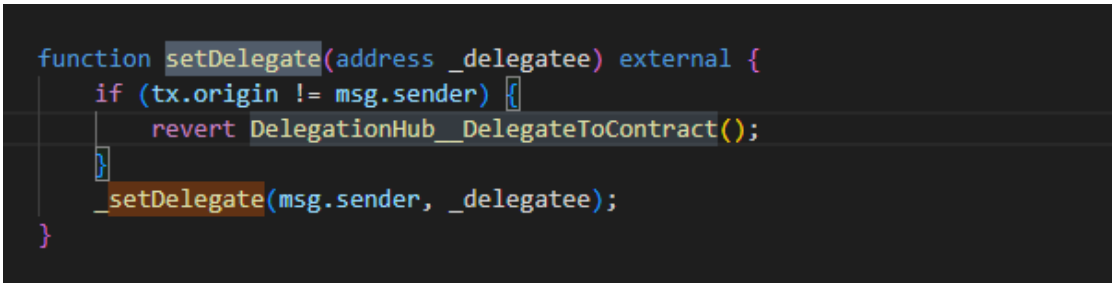
Category	Severity	Location	Status
Code Issue	Medium	DelegateHub.sol:86	**Solved**

Description

The binding mechanism in the setDelegate function of DelegateHub is insecure and vulnerable to phishing and multiple binding issues. This is because the uniqueness of the address is not verified, allowing a malicious actor to set different users to the same 1ct address. As a result, they can manipulate other users' accounts and steal their assets.

Vulnerability Analysis

1. In the DelegateHub contract, using the setDelegate function to set the same delegate address



If such a scenario exists where:

User 0xA sets the delegate address 1ct as 0xax.

Malicious user 0xB also sets the delegate address 1ct as 0xax (the

same address as user 0xA's 1ct address).

```
function _setDelegate(address _delegator, address _delegatee) internal {
    delegations[_delegator] = _delegatee;
    emit SetDelegate(_delegator, _delegatee);
}
```

2. Calling the tradeDelegate function under the circumstances where:

   delegations[0xA] == 0xax (where 0xax is the delegate address associated with user 0xA's 1ct)

   delegations[0xB] == 0xax (where 0xax is the delegate address associated with user 0xB's 1ct)

   and bypassing the permission check delegations[trader] != msg.sender allows user 0xA to set the trader parameter as user 0xB's address, thereby gaining unauthorized access to user 0xB's account information.

```
function traderDelegate(address trader, Call[] calldata calls) external payable returns (Result[] memory returnData) {
    if (delegations[trader] != msg.sender) {
        revert DelegationHub__Unauthoried();
    }
    returnData = _aggregate(trader, calls);
}
```

Recommendation

1. Ensure the uniqueness of 1CT addresses and user addresses, and prohibit multiple bindings to prevent unauthorized operations.
2. Strengthen the security of the initial binding process by implementing specific restrictions. For example, require users to stake a certain amount of tokens or perform additional actions when binding to a 1CT address. This helps prevent attackers from

misleading users into binding to malicious 1CT addresses.

SSX-03(Medium):x.Delegatehub Arbitrary External Call

Category	Severity	Location	Status
Code Issue	Medium	DelegateHub.sol:128	**Solved**

Description

There is an Arbitrary External Call vulnerability in the _aggregate method of the DelegateHub contract, allowing an attacker to exploit it by calling the approve and transfer methods of WETH to steal users' WETH assets transferred to the Hub contract.

Vulnerability Analysis

1. There is an Arbitrary External Call issue present in this scenario. A malicious attacker can exploit it by using the Hub contract to call the approve method of WETH and grant an unlimited approval to a malicious address.

```
function _aggregate(address sender, Call[] calldata calls) internal returns (Result[] memory returnData) {
    if (senderOverride != address(0)) {
        revert DelegationHub__ReentrantCall();
    }
    senderOverride = sender;
    uint256 valAccumulator;
    uint256 length = calls.length;
    returnData = new Result[](length);
    Call calldata calli;
    for (uint256 i; i < length; ) {
        Result memory result = returnData[i];
        calli = calls[i];
        uint256 val = calli.value;
        unchecked {
            valAccumulator += val;
        }
        (bool success, bytes memory retData) = calli.target.call{value: val}(calli.payload);
        if (!success && !calli.allowFailure) {
            if (retData.length == 0) revert();
            assembly {
                revert(add(0x20, retData), mload(retData))
            }
        } else {
            result.success = success;
            result.returnData = retData;
        }
        unchecked {
            ++i;
        }
    }
    if ((msg.value + receivedEth) != valAccumulator) {
        revert DelegationHub__ValueMismatch();
    }
    receivedEth = 0;
    senderOverride = address(0);
}

function _authorizeUpgrade(address newImplementation) internal override onlyOwner {}
}
```

2. When other legitimate users call the corresponding method in the UB contract through the Hub contract, the WETH token balance of the Hub contract will increase.

```
157     function withdrawEthForFee(uint256 _amount) external whenNotPaused {
158         address user = msgSender();
159         _checkBlacklist(user);
160         userBalance[address(weth)][user] -= _amount;
161         weth.withdraw(_amount);
162         IDelegationHub(hub).receiveEthFromUserBalance{value: _amount}();
163         emit Withdraw(user, address(weth), _amount);
164         _emitUserBalanceUpdate(user, address(weth));
165     }
166
```

3. When a malicious attacker confirms the presence of WETH in the Hub contract address, they can proceed to exploit it by using the transfer function to steal users' WETH tokens.

Recommendation

It is recommended to establish a mapping to maintain a record of target addresses and the corresponding methods that can be called on those addresses.

SSX-04(Low): x.Delegatehub TraderDelegate Zero Address Bypass

Category	Severity	Location	Status
Code Issue	Medium	DelegateHub.sol:86	**Solved**

Description

In the traderDelegate method of DelegateHub, the condition if (delegations[trader] != msg.sender) is used to check if a user has authorized the "trader" as their delegate address. However, if a user has not set a delegate address, the value of delegations[trader] will be 0. This means that if msg.sender is the zero address, it can bypass the condition check.

This issue cannot be exploited on the Ethereum mainnet, but it may exist on derivative chains or other Ethereum-based networks where certain system transactions use the zero address. Such as this transaction:

https://arbiscan.io/tx/0x68fbb0145741531ab272d92f34fd9b1b6df62186a3f3a444f17471231146c07e

Damocles

Transaction Hash:	0x68fbb0145741531ab272d92f34fd9b1b6df62186a3f3a444f17471231146c07e
Status:	Success
Block:	130770594 404747 L1 Block Confirmations
Timestamp:	56 days 16 hrs ago (Sep-13-2023 04:40:08 PM +UTC)
Transaction Action:	Transfer 0.01 ETH To 0x05a131e657855263da...
From:	0x0000000000000000000000000000000000000000 (Null: 0x000...000)
To:	0x05a131e657855263da33856df61ac3f02a67190b
Value:	0.01 ETH ($19.15)
Transaction Fee:	0 ETH ($0.00)
Gas Price Bid:	0 ETH (0 ETH)
Gas Price Paid:	0.0000000001 ETH (0.1 Gwei)
Ether Price:	$1,607.99 / ETH
Gas Limit & Usage by Txn:	0 | 0 (0%)
Other Attributes:	Txn Type: 100 (Deposit) Nonce: 0 Position In Block: 1
Input Data:	0x

Vulnerability Analysis

If a user has not set a delegate address, the value of delegations[trader] will default to the zero address. In this case, if msg.sender is also set to the zero address, it can bypass the permission check and potentially manipulate arbitrary user data.

```
function traderDelegate(address trader, Call[] calldata calls) external payable returns (Result[] memory returnData) {
    if (delegations[trader] != msg.sender) {
        revert DelegationHub__Unauthoried();
    }
    returnData = _aggregate(trader, calls);
}
```

Recommendation

Add an additional check to verify if msg.sender is not the zero address (0x0). .

About

Damocles is a 2023 web3 security company specializing in online security services, including smart contract audit, Product audit, penetration testing, GameFi security audit and cheat detection.

Main Web: https://damocleslabs.com/

Medium: https://damocleslabs.medium.com/

Twitter: https://twitter.com/DamoclesLabs

Email: support@damocleslabs.com