Competitive Security Assessment

DeShilling

Jul 27th, 2023

Secure3

secure3.io

Summary 3

Overview 4

Audit Scope 5

Code Assessment Findings 6

DES-1:round's `investedEth` or `investedToken` can be manipulated by the vendor 8

DES-2: `round.maxToken` can be exceeded in `ReferralInvestment::invest()` 13

DES-3:Use `safeTransferFrom` function instead of TransferFrom 16

DES-4:Anyone can withdraw the ETH in `ReferralInvestment` 17

DES-5: Privilege can't transfer to another account 19

DES-6:Gas Optimization: use `break` to jump out the loop ranther than set `i` 20

DES-7:Gas Optimization: Cache array length outside for loop 21

DES-8:Redundant code in import 22

Disclaimer 23

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	DeShilling
Platform & Language	Solidity
Codebase	• https://github.com/StartfundInc/Des-Referral • audit commit - Code shared by zip file • final commit - f958986c2a1d1e445bf9cc475bb4e5ddcf45ab13
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

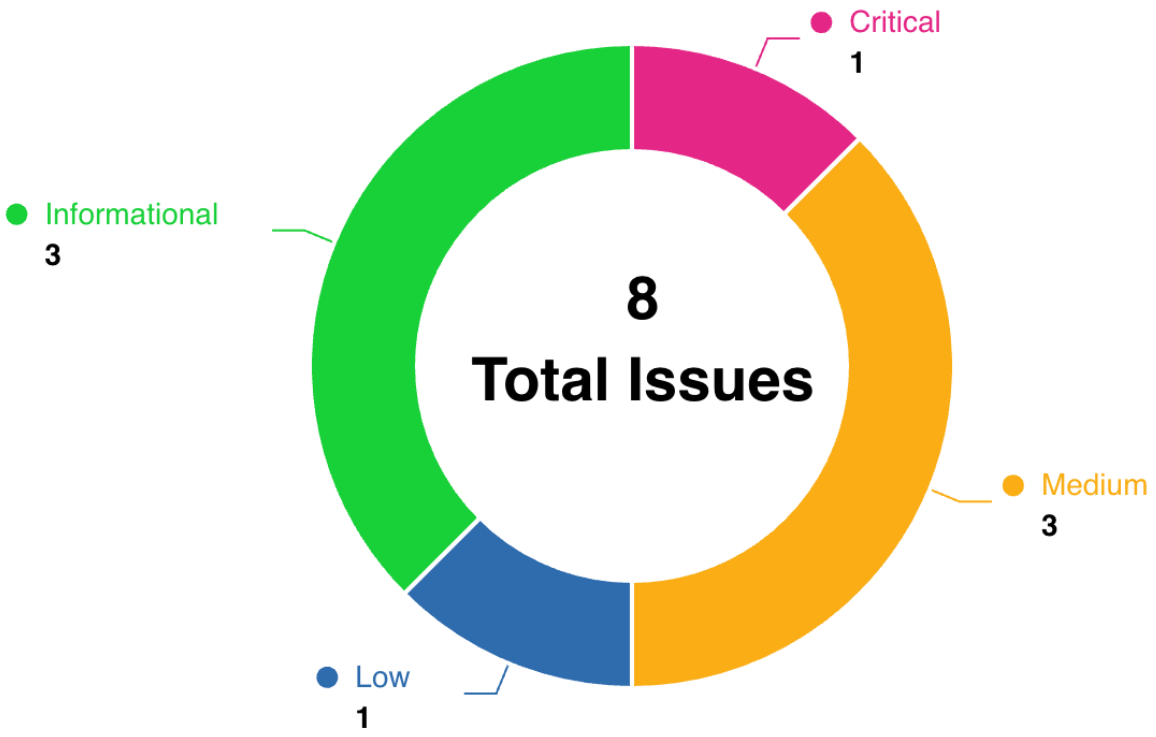
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	1	0	0	1	0	0
Medium	3	0	1	2	0	0
Low	1	0	1	0	0	0
Informational	3	0	1	2	0	0

Audit Scope

File	SHA256 Hash
./deShilling_3.3_v2.sol	ec6c931aea96f05913777888a4274f5b2113c145c20a476ee4b0cc1754d0e244

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
DES-1	round's `investedEth` or `investedToken` can be manipulated by the vendor	Logical	Critical	Fixed	Secure3
DES-2	`round.maxToken` can be exceeded in `ReferralInvestment::invest()`	Logical	Medium	Fixed	Secure3
DES-3	Use `safeTransferFrom` function instead of TransferFrom	Logical	Medium	Fixed	Secure3

DES-4	Anyone can withdraw the ETH in `ReferralInvestment`	Logical	Medium	Acknowledged	Secure3
DES-5	Privilege can't transfer to another account	Privilege Related	Low	Acknowledged	Secure3
DES-6	Gas Optimization: use `break` to jump out the loop ranther than set `i`	Gas Optimization	Informational	Fixed	Secure3
DES-7	Gas Optimization: Cache array length outside for loop	Gas Optimization	Informational	Fixed	Secure3
DES-8	Redundant code in import	Gas Optimization	Informational	Acknowledged	Secure3

DES-1:round's `investedEth` or `investedToken` can be manipulated by the vendor

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	Secure3

Code Reference

- code/deShilling_3.3_v2.sol#L62-L109

```
62:         require(campaigns[_campaignId].vendor != address(0), "Campaign does not exist.");
63:
64:         bool pass = false;
65:         uint256 ind = 0;
66:         for(uint i=0; i= _round.startDate && block.timestamp <= _round.endDate){
69:                 pass = true;
70:                 ind = i;
71:                 i = campaigns[_campaignId].rounds.length;
72:             }
73:         }
74:         require (pass, "No Active Round of Funding.");
75:         Round memory round = campaigns[_campaignId].rounds[ind];
76:         address payable campaignW = payable(campaigns[_campaignId].campaignWallet);
77:         if (msg.value > 0){
78:             require (round.investedEth + msg.value < round.maxEth || round.maxToken == 0, "Max In
vestment Reached.");
79:             if (_referralAddress != address(0)){
80:                 uint256 toCampaignWallet = msg.value*campaigns[_campaignId].fee/(10**8);
81:                 uint256 toVendor = msg.value - toCampaignWallet;
82:                 campaignW.transfer(toCampaignWallet);
83:                 payable(campaigns[_campaignId].vendor).transfer(toVendor);
84:             }
85:             else{
86:                 payable(campaigns[_campaignId].vendor).transfer(msg.value);
87:             }
88:             campaigns[_campaignId].rounds[ind].investedEth += msg.value;
89:             emit investmentETH(msg.sender, msg.value, _referralAddress, _referralId);
90:         }
91:         else if (_amount > 0){
92:             require (round.investedToken + msg.value < round.maxToken || round.maxToken == 0, "Ma
x Investment Reached.");
93:             if (_referralAddress != address(0)){
94:                 uint256 toCampaignWallet = _amount*campaigns[_campaignId].fee/(10**8);
95:                 uint256 toVendor = _amount - toCampaignWallet;
96:                 ERC20(token).transferFrom(msg.sender, campaignW, toCampaignWallet);
97:                 ERC20(token).transferFrom(msg.sender, campaigns[_campaignId].vendor, toVendor);
98:             }
99:             else{
100:                 ERC20(token).transferFrom(msg.sender, campaigns[_campaignId].vendor,
```

```
_amount);
101:            }
102:            campaigns[_campaignId].rounds[ind].investedToken += _amount;
103:            emit investmentERC20(msg.sender, _amount, _referralAddress, _referralId);
104:        }
105:    }
106:
107:    function changeCampaignWallet(uint256 _campaignId, address _campaignWallet) public {
108:        require (msg.sender == admin || msg.sender == campaigns[_campaignId].campaignWallet, "On
ly Admin can Access");
109:        campaigns[_campaignId].campaignWallet = _campaignWallet;
```

Description

Secure3 : Normally vendor can start campaign by calling the `startCampaign()` function with proper argument, however the contract cannot check whether the `_campaignWallet` is controlled by the vendor(in almost all the cases vendor do control the `_campaignWallet`).

Let's take depositing token as an example(depositing eth has the problem too).First let's assume that the campaign's fee is `10**7`(could be any value, even vendor can set it to 0).The vendor first invest `10` token to the campaign using his `CampaignWallet` address with a valid `_referralAddress`, then the contract will take `10 * 10**7/10**8`(1) token to the `CampaignWallet`, and the rest(9) to the `Vendor`. After all that the `10` amount is added to the `investedToken`.

You can see that during the whole operation the `10` token just flow from vendor to himself, the vendor can just transfer 9 token to `CampaignWallet` and perform that action again and again. He lose nothing but the `investedToken` is increased.Below here is a hardhat poc script:

```
// We require the Hardhat Runtime Environment explicitly here. This is optional
// but useful for running the script in a standalone fashion through `node <script>`.
//
// You can also run a script with `npx hardhat run <script>`. If you do that, Hardhat
// will compile your contracts, add the Hardhat Runtime Environment's members to the
// global scope, and execute the script.
const hre = require("hardhat");

async function main() {
  const nowDate = Math.round(Date.now() / 1000);
  const startDate = nowDate - 600;
  const endDate = nowDate + 600;
  const maxEth = ethers.MaxUint256;
  const maxToken = ethers.MaxUint256;
  const [owner, vendor, campaignWallet, maliciousWallet, referral] = await ethers.getSigners();
  const rounds = [[startDate, endDate, maxEth, 0, maxToken, 0]];
  const investAmount = hre.ethers.parseEther("10");
  const fee = 10**7;
  const vendorGet = (1 - (10**7/10**8)) * 10;
  const vendorGetAmount = hre.ethers.parseEther(vendorGet.toString());

  const WETH_F = await ethers.getContractFactory("WETH9");
  const WETH = await WETH_F.connect(owner).deploy();
  await WETH.waitForDeployment();
  const ReferralInvestment_F = await ethers.getContractFactory("ReferralInvestment");
  const ReferralInvestment = await ReferralInvestment_F.deploy(WETH.getAddress());
  await ReferralInvestment.waitForDeployment();

  await ReferralInvestment.connect(vendor).startCampaign(0, fee, vendor, maliciousWallet, rounds);
  await WETH.connect(vendor).approve(maliciousWallet, ethers.MaxUint256);
  await WETH.connect(maliciousWallet).deposit({ value: investAmount });
  console.log("user balance before: ", await WETH.balanceOf(maliciousWallet));
  await WETH.connect(maliciousWallet).approve(ReferralInvestment.getAddress(), ethers.MaxUint256)

  for (var i = 0; i < 10; i++) {
    await ReferralInvestment.connect(maliciousWallet).invest(investAmount, 0, referral, 0);
    await WETH.connect(maliciousWallet).transferFrom(vendor, maliciousWallet, vendorGetAmount);
  }
```

```
//   function getRoundinvestedToken(uint256 campaignsId, uint256 roundId) public view returns(uint25
6){
//     return campaigns[campaignsId].rounds[roundId].investedToken;
// }
  console.log("total investedToken: ", await ReferralInvestment.getRoundinvestedToken(0, 0));
  console.log("users balance after: ", await WETH.balanceOf(maliciousWallet));
}

// We recommend this pattern to be able to use async/await everywhere
// and properly handle errors.
main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

This vulnerability can cause the `investedToken` easily manipulated by the vendor, and if the number is used on the webapp or something else, it can trick other investors to invest by making them think that this campaign is promising and a lot of people have invested.

Recommendation

Secure3 : Make sure the `_campaignWallet` is frozen during the whole campaign.

Client Response

Fixed, There is no motivation for the Vendor to do that. Also, the Campaign Wallet is controlled by a separate Admin party, not the Vendor.

DES-2: `round.maxToken` can be exceeded in `ReferralInvestment::invest()`

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	Secure3

Code Reference

- code/deShilling_3.3_v2.sol#L77-L103

```
77:         if (msg.value > 0){
78:             require (round.investedEth + msg.value < round.maxEth || round.maxToken == 0, "Max In
vestment Reached.");
79:             if (_referralAddress != address(0)){
80:                 uint256 toCampaignWallet = msg.value*campaigns[_campaignId].fee/(10**8);
81:                 uint256 toVendor = msg.value - toCampaignWallet;
82:                 campaignW.transfer(toCampaignWallet);
83:                 payable(campaigns[_campaignId].vendor).transfer(toVendor);
84:             }
85:             else{
86:                 payable(campaigns[_campaignId].vendor).transfer(msg.value);
87:             }
88:             campaigns[_campaignId].rounds[ind].investedEth += msg.value;
89:             emit investmentETH(msg.sender, msg.value, _referralAddress, _referralId);
90:         }
91:         else if (_amount > 0){
92:             require (round.investedToken + msg.value < round.maxToken || round.maxToken == 0, "Ma
x Investment Reached.");
93:             if (_referralAddress != address(0)){
94:                 uint256 toCampaignWallet = _amount*campaigns[_campaignId].fee/(10**8);
95:                 uint256 toVendor = _amount - toCampaignWallet;
96:                 ERC20(token).transferFrom(msg.sender, campaignW, toCampaignWallet);
97:                 ERC20(token).transferFrom(msg.sender, campaigns[_campaignId].vendor, toVendor);
98:             }
99:             else{
100:                 ERC20(token).transferFrom(msg.sender, campaigns[_campaignId].vendor, _amount);
101:              }
102:              campaigns[_campaignId].rounds[ind].investedToken += _amount;
103:              emit investmentERC20(msg.sender, _amount, _referralAddress, _referralId);
```

Description

Secure3 : In `invest()` function, it perform different logical base on the `msg.value` and `_amount`. Note that if an investor call this function with `_amount` greater than zero and `msg.value` = 0, the `if` statement will skip the first block of code and go to the second block, then it will perform `require (round.investedToken + msg.value < round.maxToken || round.maxToken == 0, "Max Investment Reached.")`.Howerer, here the `msg.value`

should be `_amount`, because if the control flow can execute to this line of code, the `msg.value` must be zero, otherwise during the last invest, the `maxToken` can be exceeded.

There is a similar problem in `if (msg.value > 0)` too, I believe the right check should be `require (round.investedEth + msg.value < round.maxEth || round.maxEth == 0, "Max Investment Reached.");` So the investor will not be limited when the campaign has no limit of eth, that is `round.maxEth = 0`

Recommendation

Secure3 : Change `require (round.investedToken + msg.value < round.maxToken || round.maxToken == 0, "Max Investment Reached.");` to `require (round.investedToken + _amount < round.maxToken || round.maxToken == 0, "Max Investment Reached.");`

and `require (round.investedEth + msg.value < round.maxEth || round.maxToken == 0, "Max Investment Reached.");` to `require (round.investedEth + msg.value < round.maxEth || round.maxEth == 0, "Max Investment Reached.");`

Client Response

Fixed

DES-3:Use `safeTransferFrom` function instead of TransferFrom

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	Secure3

Code Reference

- code/deShilling_3.3_v2.sol#L122

```
122:            ERC20(token).transferFrom(msg.sender, earningWallet, _earnings[i].amount);
```

Description

Secure3 : The TransferFrom and Transfer functions do not return a bool value which can be used to judge if the call is successful.Some tokens do not revert if the transfer failed but return false instead.

Recommendation

Secure3 : Recommend using OpenZeppelin's SafeERC20 versions with the safeTransfer functions that handle the return value check as well as non-standard-compliant tokens.

Client Response

Fixed

DES-4:Anyone can withdraw the ETH in `ReferralInvestment`

Category	Severity	Client Response	Contributor
Logical	Medium	Acknowledged	Secure3

Code Reference

- code/deShilling_3.3_v2.sol#L112-L117

```
112:    function distribute(Earning[] memory _earnings) public payable {
113:        for(uint i=0; i<_earnings.length; i++){
114:            address payable earningWallet = payable(_earnings[i].wallet);
115:            earningWallet.transfer(_earnings[i].amount);
116:        }
117:    }
```

Description

Secure3 :

```
function distribute(Earning[] memory _earnings) public payable {
        for(uint i=0; i<_earnings.length; i++){
            address payable earningWallet = payable(_earnings[i].wallet);
            earningWallet.transfer(_earnings[i].amount);
        }
    }
```

If someone accidentally transferred the ETH to `ReferralInvestment` contracts, anyone can call `distribute` function to withdraw it.

Recommendation

Secure3 :

```
function distribute(Earning[] memory _earnings) public payable  {
    require(msg.sender == admin);
    ....
```

Client Response

Acknowledged, This is a weakness in the contract. Although in this flow, there will never be ETH in the contract to distribute. Will remove the weakness in next patch.

DES-5: Privilege can't transfer to another account

Category	Severity	Client Response	Contributor
Privilege Related	Low	Acknowledged	Secure3

Code Reference

- code/deShilling_3.3_v2.sol#L39

```
39:    address admin;
```

Description

Secure3 : The `admin` only set in constructor, but there are no functions to change or transfer privileges.

Recommendation

Secure3 : Recommend adding 2 steps logic, that is similar to acceptOwnership function in the Ownable2Step.sol contract to ensure that the new owner has the power to choose to become the owner of the new wallet or not. Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol

Client Response

Acknowledged, By Design.

DES-6:Gas Optimization: use `break` to jump out the loop ranther than set `i`

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	Secure3

Code Reference

- code/deShilling_3.3_v2.sol#L66-L73

```
66:         for(uint i=0; i< _round.startDate && block.timestamp <= _round.endDate){
69:                 pass = true;
70:                 ind = i;
71:                 i = campaigns[_campaignId].rounds.length;
72:             }
73:         }
```

Description

Secure3 : In `invest()` function, it use a for loop to find the suitable round, once the `pass` is set to `true`, it will jump out the loop.The problem is here in order to jump out, developer choose to set `i = campaigns[_campaignId].rounds.length;`, actually he can use just a `break` statement to achieve the same result and will save some gas too.

Recommendation

Secure3 : change the code to this:

```
    for(uint i=0; i<campaigns[_campaignId].rounds.length; i++){
                Round memory _round = campaigns[_campaignId].rounds[i];
                if (block.timestamp >= _round.startDate && block.timestamp <= _round.endDate){
                    pass = true;
                    ind = i;
                    break;
                }
            }
```

Client Response

Fixed

DES-7:Gas Optimization: Cache array length outside for loop

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	Secure3

Code Reference

- code/deShilling_3.3_v2.sol#L113
- code/deShilling_3.3_v2.sol#L120

```
113:        for(uint i=0; i<_earnings.length; i++){

120:        for(uint i=0; i<_earnings.length; i++){
```

Description

Secure3 : Caching the array length outside a loop saves reading it on each iteration

Recommendation

Secure3 :

```
+ uint length = arr.length
- for (uint256 i = 0; i < arr.length; i++) {
+ for (uint256 i = 0; i < lenth; i++) {
    // invariant: array's length is not changed
}
```

Client Response

Fixed

DES-8:Redundant code in import

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	Secure3

Code Reference

- code/deShilling_3.3_v2.sol#L4

```
4:import 'hardhat/console.sol';
```

Description

Secure3 : During the testing process, some test dependencies may be introduced, such as hardhat/console.sol, which should be removed before deployment.

Recommendation

Secure3 : remove `import 'hardhat/console.sol';`

Client Response

Acknowledged

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3's prior written consent in each instance.

This report is not an "endorsement" or "disapproval" of any particular project or team. This report is not an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3's position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.