Protocol Audit Report

Version 1.0

IvanOnchain

August 26, 2024

Steaking Audit Report

IvanOnchain

August 26, 2024

Prepared by: IvanOnChain Lead Auditors: - xxxxxxx

Table of Contents

Protocol Summary

Steak is a yield farming protocol in its pre-launch phase. It boasts an attractive APY, various vault management strategies, and a strong and active community. Being in the pre-launch phase, Steak wants to bootstrap liquidity for its ERC4626 WETH vault and reward early adopters. For this, Steak has launched a points campaign where users can stake their ETH and earn points, which will allow users to be eligible for the $STEAK token airdrop in the future.

The staking period lasts for a total of 4 weeks where users can stake their raw ETH in the `Steaking` contract. The minimum amount that can be staked is `0.5` ether. `1` ETH staked gives the user 1000 points on the backend server. Users can unstake to adjust their staked ETH amount, or withdraw it completely.

After the 4 week staking period ends, the Steak protocol team will set the address of the freshly deployed ERC4626 WETH vault. Users will be able to convert their raw staked ETH into WETH, deposit into the WETH vault, and claim their shares.

Disclaimer

IvanOnChain makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity/Vyper implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Scope

All the files listed below are in scope.

```
src
 -- steaking-contracts
|    -- src
|         -- Steaking.vy
 -- steaking-server
     -- src
          -- models
         |     -- steakPoints.js
          -- utils
         |     -- connectToMongoDb.js
         |     -- constants.js
         |     -- getConfig.js
          -- main.js
```

Roles

1. **Users**: Can stake and unstake raw ETH into the vault. After the staking period ends, users can convert ETH to WETH, and deposit it into the WETH vault.
2. **Steak protocol team multisig**: The multisig is the owner of the `Steaking` contract, and is responsible for setting the vault address after the staking period ends.

Issues found

Severity	Number of issues found
High	3
Medium	0
Low	1
Info	0
Total	4

Findings

High

[H-1] `Steaking::Stake` function doesn't add up the user staked amount

Description

Users should be able to increase their total staked amount every time they stake a new amount, but the `Steaking::usersToStakes` state variable will reflect the last user staked amount and the accumulated.

Impact

User will expect to increase their total staked amount to have more ETH to deposit into the vault but at the end just will able to deposit the last staked amount because `Steaking::Stake` doesn't add up the value.

```
    @external
    @payable
    def stake(_onBehalfOf: address):
        """
        @notice Allows users to stake ETH for themselves or any other
            user within the staking period.
        @param _onBehalfOf The address to stake on behalf of.
        """
        assert not self._hasStakingPeriodEnded(),
            STEAK__STAKING_PERIOD_ENDED
        assert msg.value >= MIN_STAKE_AMOUNT,
            STEAK__INSUFFICIENT_STAKE_AMOUNT
        assert _onBehalfOf != ADDRESS_ZERO, STEAK__ADDRESS_ZERO

    @>  self.usersToStakes[_onBehalfOf] = msg.value
        self.totalAmountStaked += msg.value

        log Staked(msg.sender, msg.value, _onBehalfOf)
```

Proof of Concepts Put next snippet code into `Steaking.t.sol` file. This test proof that the final `usersToStake` amount is not the total amount staked by the user.

```

function testStakedAmountDoesNotAccumulative() public {
        uint256 dealAmount = steaking.getMinimumStakingAmount();
        vm.deal(attacker, dealAmount);
        uint16 numberOfStakes = 3;

        for (uint16 i = 0; i < numberOfStakes; i++) {
            _stake(user1, dealAmount, user1);
        }

        assertEq(steaking.usersToStakes(user1), dealAmount);
    }
```

Recommended mitigation

```
    @external
    @payable
    def stake(_onBehalfOf: address):
        """
        @notice Allows users to stake ETH for themselves or any other
            user within the staking period.
        @param _onBehalfOf The address to stake on behalf of.
        """
        assert not self._hasStakingPeriodEnded(),
            STEAK__STAKING_PERIOD_ENDED
        assert msg.value >= MIN_STAKE_AMOUNT,
            STEAK__INSUFFICIENT_STAKE_AMOUNT
        assert _onBehalfOf != ADDRESS_ZERO, STEAK__ADDRESS_ZERO

+       self.usersToStakes[_onBehalfOf] += msg.value
-       self.usersToStakes[_onBehalfOf] = msg.value
        self.totalAmountStaked += msg.value

        log Staked(msg.sender, msg.value, _onBehalfOf)
```

[H-2] An attacker could use other people's funds to deposit into the vault in their favor.

Description The `Steaking::depositIntoVault` function doesn't reduce the stake balance when a user deposit so this doesn't avoid the user call again the function if the contract has more balance.

Impact

An attacker can user vault balance in their favor to deposit into the vault.

Proof of Concepts

Copy this code snippet into `Steaking.t.sol` file.

```
    function testCanDepositToVaultBalanceFromOtherUser() public {
        uint256 dealAmount = steaking.getMinimumStakingAmount();
        _stake(user1, dealAmount, user1);
        _stake(attacker, dealAmount, attacker);

        _endStakingPeriod();

        vm.startPrank(owner);
        steaking.setVaultAddress(address(wethSteakVault));
        vm.stopPrank();

        vm.startPrank(attacker);
        steaking.depositIntoVault();
        steaking.depositIntoVault();
        vm.stopPrank();

        vm.startPrank(user1);
        // It should revert because of OutOfFunds error
        vm.expectRevert();
        steaking.depositIntoVault();
        vm.stopPrank();

        // attacker wethSteakVault balance should be its balance plus
           user1 balance.
        assertEq(wethSteakVault.balanceOf(attacker), dealAmount * 2);
    }
```

Recommended mitigation

```
    @external
    def depositIntoVault() -> uint256:
        """
        @notice Allows users who have staked ETH during the staking
           period to deposit their ETH
        into the WETH Steak vault.
        @dev Before depositing into the vault, the raw ETH is converted
            into WETH.
        @return The amount of shares received from the WETH Steak vault
           .
        """
        assert self._hasStakingPeriodEndedAndVaultAddressSet(),
           STEAK__STAKING_PERIOD_NOT_ENDED_OR_VAULT_ADDRESS_NOT_SET

        # q user stake amount shouldn't be reduced?

        stakedAmount: uint256 = self.usersToStakes[msg.sender]
+       self.usersToStakes[msg.sender] -= stakedAmount
+       self.totalAmountStaked -= stakedAmount

        assert stakedAmount > 0, STEAK__AMOUNT_ZERO

        extcall IWETH(WETH).deposit(value=stakedAmount)
        extcall IWETH(WETH).approve(self.vault, stakedAmount)
        sharesReceived: uint256 = extcall IWETHSteakVault(self.vault).
           deposit(stakedAmount, msg.sender)

        log DepositedIntoVault(msg.sender, stakedAmount, sharesReceived
           )
```

```

            return sharesReceived
```

[H-3] Backend server does not take into account unstake amounts to reduce user points.

Description

The backend server only listens to one specific event and does not track unstake events. As a result, when someone unstakes, it does not impact the points calculation.

Impact

A user can repeatedly stake and unstake to artificially inflate their awarded points.

Proof of Concepts

Backend sever only listen `Stake` events, therefore doesn't way to reduce the points balance if somebody unstake.

Recommended mitigation

Add listener for `Unstake` event and add logic to reduce the points balance.

Medium

Low

[L-1] Risk of blocked funds if it is not possible set the vault address.

Description `Steaking` contract only allow to withdraw funds before staking period ends, after it the only way to get the funds back is through the vaults. However if for any reason the owner is unable to set the vaults address, funds will be blocked for ever,

Impact

If the owner dies, loses the key to sign transactions, or for some reason is unable to establish the vault address, users will lose access to their funds.

Proof of Concepts

`Steaking::unstake` function has a requirement that stablish that only is possible unstake before staking period ends.

```
@external
def unstake(_amount: uint256, _to: address):
    """
    @notice Allows users to unstake their staked ETH before the staking period ends. Users
    can adjust their staking amounts to their liking.
    @param _amount The amount of staked ETH to withdraw.
    @param _to The address to send the withdrawn ETH to.
    """

    @> assert not self._hasStakingPeriodEnded(), STEAK__STAKING_PERIOD_ENDED
    assert _to != ADDRESS_ZERO, STEAK__ADDRESS_ZERO

    stakedAmount: uint256 = self.usersToStakes[msg.sender]
    assert stakedAmount > 0 and _amount > 0, STEAK__AMOUNT_ZERO
    assert _amount <= stakedAmount, STEAK__INSUFFICIENT_STAKE_AMOUNT

    self.usersToStakes[msg.sender] -= _amount
    self.totalAmountStaked -= _amount

    send(_to, _amount)

    log Unstaked(msg.sender, _amount, _to)
```

`Steaking::depositIntoVault` function only allows to deposit if the vaults address is set previously.

```
@external
def depositIntoVault() -> uint256:
    """
    @notice Allows users who have staked ETH during the staking period to deposit their ETH
    into the WETH Steak vault.
    @dev Before depositing into the vault, the raw ETH is converted into WETH.
    @return The amount of shares received from the WETH Steak vault.
    """
@>  assert self._hasStakingPeriodEndedAndVaultAddressSet(),

   STEAK__STAKING_PERIOD_NOT_ENDED_OR_VAULT_ADDRESS_NOT_SET

    stakedAmount: uint256 = self.usersToStakes[msg.sender]
    assert stakedAmount > 0, STEAK__AMOUNT_ZERO

    extcall IWETH(WETH).deposit(value=stakedAmount)
    extcall IWETH(WETH).approve(self.vault, stakedAmount)
    sharesReceived: uint256 = extcall IWETHSteakVault(self.vault).deposit(stakedAmount, msg.sender)

    log DepositedIntoVault(msg.sender, stakedAmount, sharesReceived)

    return sharesReceived
```

Recommended mitigation

It is recommended to add a condition in the staking function that allows users to unstake their funds if the vault address is not set within a certain period after the staking period has ended. # Informational # Gas