MythX Report
For Token Generation Event

Overview

Project Name	zkSwap Finance - TGE
Auditor	MythX.io
Source Code	https://github.com/ZkSwapFinance/zf-launchpad
Mode	Deep
Time	Sat Aug 19th 2023
Status	**Passed**

Summary

Done	Contract	Network	Low Risk Issues
☑	ZFLaunchpadNative.sol	zkSync - ETH - Arbitrum	7
☑	ZFLaunchpad.sol	BSC - Polygon	26

Reference

MythX Passed Badge on Github mainnet-contracts ZkSwap Finance Mainnet contracts MythX passed	https://github.com/ZkSwapFinance/zf-launchpad

MythX

REPORT 64E033BC21306D001AE19CD5

Created	Sat Aug 19 2023 03:15:08 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	648fc02af4bf584372592643

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
6fd98797-2e60-41b6-9c1a-602d8c5b42c6	/launchpad/zflaunchpadnative.sol	7

Analysis 6fd98797-2e60-41b6-9c1a-602d8c5b42c6 MythX

Started	Sat Aug 19 2023 03:15:14 GMT+0000 (Coordinated Universal Time)
Finished	Sat Aug 19 2023 04:00:24 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Vscode-Extension
Main Source File	/Launchpad/Zflaunchpadnative.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	7

ISSUES

LOW

SWC-107

A call to a user-supplied address is executed.

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

/launchpad/zflaunchpadnative.sol

Locations

```
81 }
82
83 function withdrawFunds() external onlyOwner {
84 uint256 amount = address(this).balance;
85 _safeTransferETH(msg.sender, amount);
86 }
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/launchpad/zflaunchpadnative.sol

Locations

```
89 // No discount
90 if (block.timestamp < startTimestamp || block.timestamp.sub(startTimestamp).div(356400) > 0) { // No bonus
91 return 0;
92 }
93 uint256 _currentTime = block.timestamp.sub(startTimestamp);
94 // Zone 3
95 if (_currentTime >= 32400 && _currentTime.sub(32400).div(324000) == 0){
96 return 15 - (_currentTime.sub(32400)).div(21600);
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/launchpad/zflaunchpadnative.sol

Locations

```
89  // No discount
90  if (block.timestamp < startTimestamp || block.timestamp.sub(startTimestamp).div(356400) > 0) { // No bonus
91  return 0;
92  }
93  uint256 _currentTime = block.timestamp.sub(startTimestamp);
94  // Zone 3
95  if (_currentTime >= 32400 && _currentTime.sub(32400).div(324000) == 0){
96  return 15 - (_currentTime.sub(32400)).div(21600);
97  }
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/launchpad/zflaunchpadnative.sol

Locations

```
42  function deposit(address _referrer) payable external nonReentrant {
43  require(block.timestamp > startTimestamp, "deposit:Too early");
44  require(block.timestamp < endTimestamp, "deposit:Too late");
45  require(isSaleStart, "deposit: Sale not yet enabled");
46
47  uint256 _amount = msg.value;
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/launchpad/zflaunchpadnative.sol

Locations

```
44  require(block.timestamp < endTimestamp, "deposit:Too late");
45  require(isSaleStart, "deposit: Sale not yet enabled");
46
47  uint256 _amount = msg.value;
48
49  require(_amount > 0, "deposit:Amount must be > 0");
50
51  UserInfo storage user = userInfo[msg.sender];
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/launchpad/zflaunchpadnative.sol

Locations

```
94  // Zone 3
95  if (_currentTime >= 32400 && _currentTime.sub(32400).div(324000) == 0){
96  return 15 - (_currentTime.sub(32400)).div(21600);
97  }
98
99  return 25 - (_currentTime.div(10800)).mul(5) + (_currentTime.div(21600)).mul(5);
100 }
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/launchpad/zflaunchpadnative.sol

Locations

```
94  // Zone 3
95  if (_currentTime >= 32400 && _currentTime.sub(32400).div(324000) == 0){
96  return 15 - (_currentTime.sub(32400)).div(21600);
97  }
98
99  return 25 - (_currentTime.div(10800)).mul(5) + (_currentTime.div(21600)).mul(5);
100 }
101
102
103 function getUserInfo(address _user) public view returns (UserInfo memory) {
104 return userInfo[_user];
105 }
```

MythX

REPORT 64E0315988F09C001AEE11DC

Created	Sat Aug 19 2023 03:04:57 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	648fc02af4bf584372592643

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
eae28395-d5cb-4c6a-b2a7-4235a184b752	/launchpad/zflaunchpad.sol	26

Analysis eae28395-d5cb-4c6a-b2a7-4235a184b752 MythX

Started	Sat Aug 19 2023 03:05:02 GMT+0000 (Coordinated Universal Time)
Finished	Sat Aug 19 2023 03:54:07 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Vscode-Extension
Main Source File	/Launchpad/Zflaunchpad.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	26

ISSUES

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
69 if (user.amount == 0) users.push(msg.sender);
70 // Set value before check discount
71 totalRaised = totalRaised.add(_amount);
72 user.amount = user.amount.add(_amount);
73
74 // Check discount
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
73
74 // Check discount
75 uint256 _bonusPercent = getBonusPercentage();
76 // Update amount + bonus
77 _amount = _amount + _amount.mul(_bonusPercent).div(100);
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
74  // Check discount
75  uint256 _bonusPercent = getBonusPercentage();
76  // Update amount + bonus
77  _amount = _amount + _amount.mul(_bonusPercent).div(100);
78  // Update user
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
96  return 0;
97  }
98  uint256 _currentTime = block.timestamp.sub(startTimestamp);
99  // Zone 3
100 if (_currentTime >= 32400 && _currentTime.sub(32400).div(324000) == 0){
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
96  return 0;
97  }
98  uint256 _currentTime = block.timestamp.sub(startTimestamp);
99  // Zone 3
100 if (_currentTime >= 32400 && _currentTime.sub(32400).div(324000) == 0){
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
81
82  // Referral
83  if (_referrer != address(0) && _referrer != msg.sender) {
84  if (referralInfo[_referrer] == 0) referrals.push(_referrer);
85  uint256 _referralAmount = _amount.mul(5).div(100);
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
82  // Referral
83  if (_referrer != address(0) && _referrer != msg.sender) {
84  if (referralInfo[_referrer] == 0) referrals.push(_referrer);
85  uint256 _referralAmount = _amount.mul(5).div(100);
86  referralInfo[_referrer] = referralInfo[_referrer].add(_referralAmount);
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
84  if (referralInfo[_referrer] == 0) referrals.push(_referrer);
85  uint256 _referralAmount = _amount.mul(5).div(100);
86  referralInfo[_referrer] = referralInfo[_referrer].add(_referralAmount);
87  totalReferralAmount = totalReferralAmount.add(_referralAmount);
88  }
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
70 // Set value before check discount
71 totalRaised = totalRaised.add(_amount);
72 user.amount = user.amount.add(_amount);
73
74 // Check discount
```

LOW

SWC-107

Write to persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
73
74 // Check discount
75 uint256 _bonusPercent = getBonusPercentage();
76 // Update amount + bonus
77 _amount = _amount + _amount.mul(_bonusPercent).div(100);
```

LOW

SWC-107

Write to persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
74 // Check discount
75 uint256 _bonusPercent = getBonusPercentage();
76 // Update amount + bonus
77 _amount = _amount + _amount.mul(_bonusPercent).div(100);
78 // Update user
79 user.amountBonus = user.amountBonus.add(_amount);
```

LOW

SWC-107

Write to persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
80 totalRaisedBonus = totalRaisedBonus.add(_amount);
81
82 // Referral
83 if (_referrer != address(0) && _referrer != msg.sender) {
84 if (referralInfo[_referrer] == 0) referrals.push(_referrer);
85 uint256 _referralAmount = _amount.mul(5).div(100);
```

LOW

SWC-107

Write to persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
81
82 // Referral
83 if (_referrer != address(0) && _referrer != msg.sender) {
84 if (referralInfo[_referrer] == 0) referrals.push(_referrer);
85 uint256 _referralAmount = _amount.mul(5).div(100);
86 referralInfo[_referrer] = referralInfo[_referrer].add(_referralAmount);
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
84 if (referralInfo[_referrer] == 0) referrals.push(_referrer);
85 uint256 _referralAmount = _amount.mul(5).div(100);
86 referralInfo[_referrer] = referralInfo[_referrer].add(_referralAmount);
87 totalReferralAmount = totalReferralAmount.add(_referralAmount);
88 }
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
86  referralInfo[_referrer] = referralInfo[_referrer].add(_referralAmount);
87  totalReferralAmount = totalReferralAmount.add(_referralAmount);
88  }
89
90  emit Deposit(msg.sender, _amount, _referrer);
91  }
```

LOW

SWC-107

Write to persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
85  uint256 _referralAmount = _amount.mul(5).div(100);
86  referralInfo[_referrer] = referralInfo[_referrer].add(_referralAmount);
87  totalReferralAmount = totalReferralAmount.add(_referralAmount);
88  }
89
90  emit Deposit(msg.sender, _amount, _referrer);
91  }
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
91  }
92
93  function getBonusPercentage() public view returns (uint256) {
94  // No discount
95  if (block.timestamp < startTimestamp || block.timestamp.sub(startTimestamp).div(356400) > 0) { // No bonus
```

LOW

SWC-107

Write to persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
88  }
89
90  emit Deposit(msg.sender, _amount, _referrer);
91  }
92
93  function getBonusPercentage() public view returns (uint256) {
94  // No discount
95  if (block.timestamp < startTimestamp || block.timestamp.sub(startTimestamp).div(356400) > 0) { // No bonus
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/launchpad/zflaunchpad.sol

Locations

```
99  // Zone 3
100 if (_currentTime >= 32400 && _currentTime.sub(32400).div(324000) == 0){
101 return 15 - (_currentTime.sub(32400)).div(21600);
102 }
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/launchpad/zflaunchpad.sol

Locations

```
94  // No discount
95  if (block.timestamp < startTimestamp || block.timestamp.sub(startTimestamp).div(356400) > 0) { // No bonus
96  return 0;
97  }
98  uint256 _currentTime = block.timestamp.sub(startTimestamp);
99  // Zone 3
100 if (_currentTime >= 32400 && _currentTime.sub(32400).div(324000) == 0){
101 return 15 - (_currentTime.sub(32400)).div(21600);
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/launchpad/zflaunchpad.sol

Locations

```
94  // No discount
95  if (block.timestamp < startTimestamp || block.timestamp.sub(startTimestamp).div(356400) > 0) { // No bonus
96  return 0;
97  }
98  uint256 _currentTime = block.timestamp.sub(startTimestamp);
99  // Zone 3
100 if (_currentTime >= 32400 && _currentTime.sub(32400).div(324000) == 0){
101 return 15 - (_currentTime.sub(32400)).div(21600);
102 }
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/launchpad/zflaunchpad.sol

Locations

```
57  function deposit(uint256 _amount, address _referrer) external nonReentrant {
58  require(block.timestamp > startTimestamp, "deposit:Too early");
59  require(block.timestamp < endTimestamp, "deposit:Too late");
60  require(isSaleStart, "deposit: Sale not yet enabled");
61
62  require(_amount > 0, "deposit:Amount must be > 0");
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/launchpad/zflaunchpad.sol

Locations

```
59  require(block.timestamp < endTimestamp, "deposit:Too late");
60  require(isSaleStart, "deposit: Sale not yet enabled");
61
62  require(_amount > 0, "deposit:Amount must be > 0");
63
64  UserInfo storage user = userInfo[msg.sender];
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/launchpad/zflaunchpad.sol

Locations

```
99  // Zone 3
100 if (_currentTime >= 32400 && _currentTime.sub(32400).div(324000) == 0){
101 return 15 - (_currentTime.sub(32400)).div(21600);
102 }
103
104 return 25 - (_currentTime.div(10800)).mul(5) + (_currentTime.div(21600)).mul(5);
105 }
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/launchpad/zflaunchpad.sol

Locations

```
99  // Zone 3
100 if (_currentTime >= 32400 && _currentTime.sub(32400).div(324000) == 0){
101 return 15 - (_currentTime.sub(32400)).div(21600);
102 }
103
104 return 25 - (_currentTime.div(10800)).mul(5) + (_currentTime.div(21600)).mul(5);
105 }
106
107
108 function getUserInfo(address _user) public view returns (UserInfo memory) {
109 return userInfo[_user];
110 }
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

/launchpad/zflaunchpad.sol

Locations

```
119 }
120
121 function getReferralInfo(address _user) public view returns (uint256) {
122 return referralInfo[_user];
123 }
```