expelee

Building the Futuristic Blockchain Ecosystem

SECURITY AUDIT REPORT

SafuStaking

TOKEN OVERVIEW

Risk Findings

Severity	Found
High	3
Medium	0
Low	0
Informational	5

Centralization Risks

Owner Privileges	Description
Can Owner Set Taxes >25% ?	Not Detected
Owner needs to enable trading ?	Not Detected
Can Owner Disable Trades ?	Not Detected
Can Owner Mint ?	Not Detected
Can Owner Blacklist ?	Not Detected
Can Owner set Max Wallet amount ?	Not Detected
Can Owner Set Max TX amount ?	Not Detected

expelee

TABLE OF CONTENTS

OVERVIEW

The Expelee team has performed a line-by-line manual analysis and automated review of the smart contract. The smart contract was analysed mainly for common smart contract vulnerabilities, exploits, and manipulation hacks. According to the smart contract audit:

Audit Result	**Passed With High Risk**
KYC Verification	**-**
Audit Date	**10 July 2023**

expelee

CONTRACT DETAILS

Token Name: SafuStaking

Network: Ethereum

Contract Address:
0x689960DC1ed2EA5C4ECd078CaF9401105c6ac89D

Owner's Wallet:
--

Deployer's Wallet:
--

Contract Type: Staking contract

Checksum: 2ed3fa52d00b75ebd6a7069b4c7c7da627c2d00f

Tests: Extensive unit tests were conducted using the forge (foundry) https://book.getfoundry.sh

AUDIT METHODOLOGY

In the process of conducting this audit, a variety of methodologies were utilized to ensure the code's reliability and functionality:

1. Manual Code Review: The codebase underwent an intensive manual review process, carried out by our experienced Expelee audit team. This meticulous, line-by-line analysis played a crucial role in identifying potential vulnerabilities, bugs, or inconsistencies that could negatively affect the code's performance.
2. Automated Testing: The deployment and functionality of the contract were tested on local test networks using automated test suites to simulate various transaction scenarios.

Despite the rigorous examination and testing methodologies implemented, we discovered several issues that led to the overall failure of this audit. These issues, as identified and discussed in our audit discussions, need to be addressed and rectified for a successful audit certification.

VULNERABILITY CHECKS

Design Logic	Passed
Compiler warnings	Passed
Private user data leaks	Passed
Timestamps dependence	Passed
Integer overflow and underflow	Passed
Race conditions & reentrancy. Cross-function race conditions	Passed
Possible delays in data delivery	Passed
Oracle calls	Passed
Front Running	Passed
DoS with Revert	Passed
DoS with block gas limit	Passed
Methods execution permissions	Passed
Economy model	Passed
Impact of the exchange rate on the logic	Passed
Malicious event log	Passed
Scoping and declarations	Passed
Uninitialized storage pointers	Passed
Arithmetic accuracy	Passed
Cross-function race conditions	Passed
Safe Zepplin module	Passed

RISK CLASSIFICATION

When performing smart contract audits, our specialists look for known vulnerabilities as well as logical and acces control issues within the code. The exploitation of these issues by malicious actors may cause serious financial damage to projects that failed to get an audit in time. We categorize these vulnerabilities by the following levels:

High Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Medium Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Low Risk

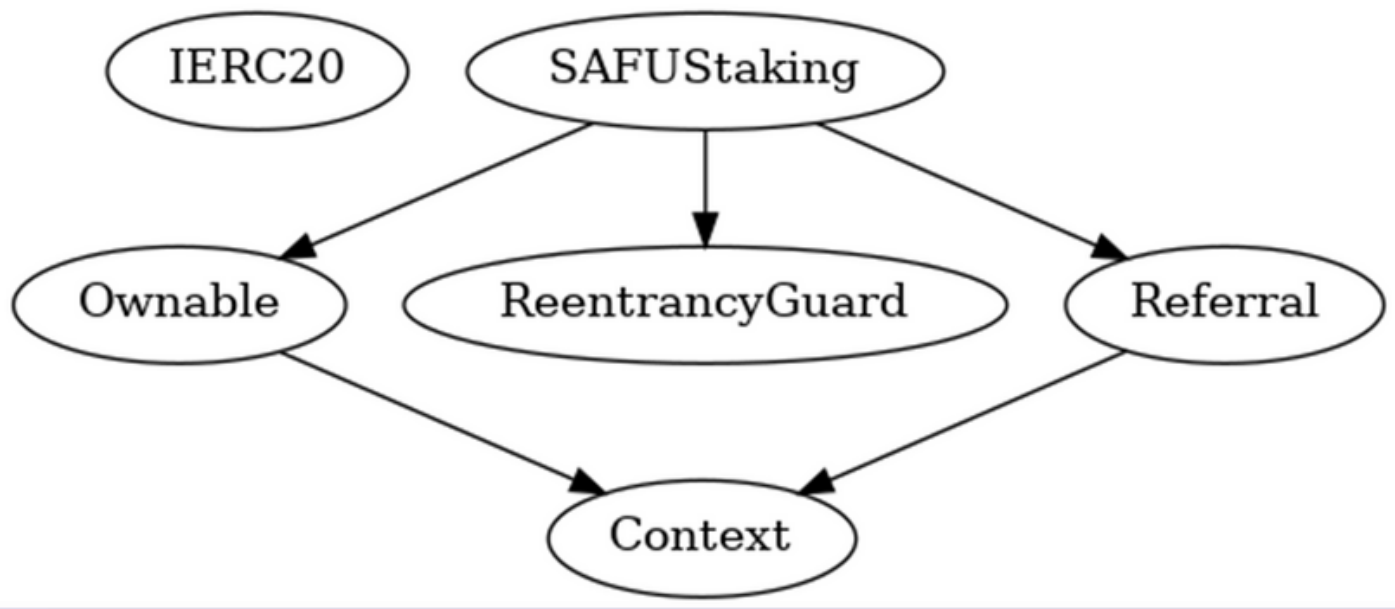
Issues on this level are minor details and warning that can remain unfixed.

Informational

Issues on this level are minor details and warning that can remain unfixed.

expelee

INHERITANCE TREES



expelee

FUNCTION DETAILS

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
└	balanceOf	External ❗	NO ❗	
└	transfer	External ❗	🛑	NO ❗
└	transferFrom	External ❗	🛑	NO ❗
Context	Implementation			
└	_msgSender	Internal 🔒		
Ownable	Implementation	Context		
└	<Constructor>	Public ❗	🛑	NO ❗
└	owner	Public ❗	NO ❗	
└	_checkOwner	Private 🔐		
└	renounceOwnership	External ❗	🛑	onlyOwner
└	transferOwnership	External ❗	🛑	onlyOwner
└	_transferOwnership	Private 🔐	🛑	
ReentrancyGuard	Implementation			
└	<Constructor>	Public ❗	🛑	NO ❗
└	_nonReentrantBefore	Private 🔐	🛑	
└	_nonReentrantAfter	Private 🔐	🛑	
└	_reentrancyGuardEntered	Private 🔐		
Referral	Implementation	Context		
└	getReferInfo	External ❗	NO ❗	
└	addReferee	Public ❗	🛑	NO ❗
└	getReferees	Public ❗	NO ❗	
SAFUStaking	Implementation	Ownable, ReentrancyGuard, Referral		
└	<Constructor>	Public ❗	🛑	NO ❗
└	deposit	External ❗	🛑	NO ❗
└	withdraw	External ❗	🛑	nonReentrant
└	emergencyWithdraw	External ❗	🛑	nonReentrant
└	claimRewards	External ❗	🛑	nonReentrant
└	pendingRewards	Public ❗	NO ❗	
└	toggleStaking	External ❗	🛑	onlyOwner
└	investorOrderIds	External ❗	NO ❗	
└	claimRefRewards	External ❗	🛑	nonReentrant
└	_calculateRefRewards	Private 🔐		
└	_sumRefRewards	Public ❗	NO ❗	
└	_totalRewards	Private 🔐		
└	getRefRewards	Public ❗	NO ❗	
└	transferAnyERC20Token	External ❗	🛑	onlyOwner

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

expelee

UNIT TESTS

Referral system:

- referral levels updated correctly (level 1 – level 5)
- referral rewards were calculated correctly (reffered total claims and pending rewards)
  - level 1 : 10%
  - level 2 : 7%
  - level 3 : 5%
  - level 4 : 4%
  - level 5 : 2%
- refferer could claim their pending tokens, which was calculated as a percentage of (total claimed + total pending rewards) of reffered user
- users are able to update their refferer which might not be in accordance with deposit function

Depositing:

- users could deposit any amount of token (except 0), the contract and user state were mostly updated correctly
- an order created with a locked amount, time and, beneficiary.

Withdraw:

- users could withdraw their locked tokens (by providing an order id) after lock time been passed.
- order pending rewards were calculated correctly and the locked amount sent to the staker.
- some states were missing to update, like stakeOnPool

Emergency withdraw:

- users could withdraw their locked tokens after any time, a 25% fee were charged and saved in the contract.
- order pending rewards were calculated correctly and the locked amount sent to the staker.

expelee

API

- during testing, we had to change visibility of some variables to “public” to be able to retrieve some variables from the contract, including but not limited to:

```
mapping(address => address[]) public referrals_level_1;
mapping(address => address[]) public referrals_level_2;
mapping(address => address[]) public referrals_level_3;
mapping(address => address[]) public referrals_level_4;
mapping(address => address[]) public referrals_level_5;
```

MANUAL REVIEW

Severity Criteria

Expelee assesses the severity of disclosed vulnerabilities according to methodology based on OWASP standarts.

Vulnerabilities are dividend into three primary risk categroies:
High
Medium
Low

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious input handling
- Escalation of privileges
- Arithmetic
- Gas use

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

HIGH RISK FINDING

Visibility – Upgrading refferer

Severity: High
Status: Not Resolved

Overview:

users are able to update their refferer using addReferee funciton. This funciton is declared as public with no access controls. Deposit function only updates refferer of the msg.sender if it doesn't have one. So addReferee being public, might not be in accordance with overall logic of the system.

Code:

```
  function deposit(uint256 _amount, uint256
_lockupDuration, address _referrer) external {
      if (_referrer != address(0) &&
!userInfo[_msgSender()].referred) {
          //@audit referee can be changed later
          addReferee(_referrer);
      }
```

HIGH RISK FINDING

```
//@audit this function should be internal? because
msg.sender can change its own refferer
 function addReferee(address ref) public {
 require(ref != _msgSender(), " You cannot refer yourself ");

 userInfo[_msgSender()].referred = true;
 userInfo[_msgSender()].referred_by = ref;
```

Suggestion:
change visibility of addReferee function to internal

HIGH RISK FINDING

Logical – Invalid refferer

Severity: High
Status: Not Resolved

Overview: users are able to add any arbitray address as their refferer even if that address did not stake in the contract before.

Code:

```
  //@audit this function should be internal? because msg.sender can change its own refferer
  function addReferee(address ref) public {
      require(ref != _msgSender(), " You cannot refer yourself ");
      userInfo[_msgSender()].referred = true;
      userInfo[_msgSender()].referred_by = ref;
```

Suggestion:
Check whether the ref have any tokens in the contract or not. This can be achieved by adding a requrie statement to check balance of “ref”, also its suggested to add a “minToStake” variable in order to be considered a valid referrer

HIGH RISK FINDING

```
function addReferee(address ref) public {
require(ref != _msgSender(), " You cannot refer yourself ");
requrie(balanceOf(ref) >= minToStake, “Invalid refferer”);
userInfo[_msgSender()].referred = true;
userInfo[_msgSender()].referred_by = ref;
```

HIGH RISK FINDING

Centralization - Ability to withdraw any ERC20 token

Severity: High
Status: Open

Overview: The contract owner is able to withdraw any ERC20 token including staked tokens from the contract. This is considered a high level centralization risk as a malicious owner is able to withdraw staked tokens of a user.

Code:

```
  function transferAnyERC20Token(address payaddress,
address tokenAddress, uint256 amount) external
onlyOwner {
IERC20(tokenAddress).transfer(payaddress, amount);
  }
```

Suggestion:

- Ensure that owner is not able to withdraw staked tokens (totalStake + totalWithdrawal)
- Implement a governance model to allow stakers for deciding whether owner can withdraw all staked tokens from the contract (e.g. in emergency situations) or not.

INFORMATIONAL FINDING

Updating – stakeOnPool not updated

Severity: Informational
Status: Open

Overview: stakeOnPool mapping stores total staked tokens in each pool. This mapping is not updated when withdrawing or emergency withdrawing funds from the contract

Suggestion:
update stakeOnPool mapping in withdraw and emergencyWithdraw functions.

INFORMATIONAL FINDING

Updating – using duration for updating rewardOnPool

Severity: Informational
Status: Open

Overview: stakeOnPool mapping uses pool identifiers (30, 180, 365) in order to update total staked tokens in each pool. But rewardOnPool is using actual durations for updating total claimed rewards of each pool (using 30 * 86400 etc)

```
rewardOnPool[orderInfo.lockupDuration] =
rewardOnPool[orderInfo.lockupDuration] +
claimAvailable;
```

Suggestion:
for API consistency, use 30, 180, 365 for updating rewardOnPool.

INFORMATIONAL FINDING

Logical – emergency withdraw

Severity: Informational
Status: Open

Overview: emergencyWithdraw function is performing all the logic of withdraw funciton plus deducting 25% fee from users. An emergencyWithdraw function should have the logic to allow users for unstaking their tokens without having to go through all the complex logic of other functions (which might be not usable due to an unkonwn bug)

Suggestion:
Change the name of emergencyWithdraw function to something like “earlyUnstake”, “exit” and create an emergencyWithdraw function to allow stakers for withdrawing their tokens freely in emergency situations without receiving rewards or paying fees.
This emergency situation can be enabled only by owner:

```
function enableEmergencyWithdraw() public onlyOwner{
        allowEmergencyWithdraw = true;
}
```

INFORMATIONAL FINDING

Suggestion – emergency withdraw deducting fees after unlock

Severity: Informational
Status: Open

Overview: emergencyWithdraw function is deducting 25% fee from users even if order is unlocked

Suggestion:
Its suggested to only deduct 25% fee when order is still locked.

INFORMATIONAL FINDING

Optimization – constant variables

Impact: Low
Status: Not Resolved

Overview: some variables are immutable and constantly being used. Its suggested to change this vairables to constant.

Suggestion:
Change below variables to constant:
uint256 private _30daysPercentage = 15;
uint256 private _180daysPercentage = 35;
uint256 private _365daysPercentage = 100;

ABOUT EXPELEE

Expelee is a product-based aspirational Web3 start-up. Coping up with numerous solutions for blockchain security and constructing a Web3 ecosystem from deal making platform to developer hosting open platform, while also developing our own commercial and sustainable blockchain.

www.expelee.com

expeleeofficial
expelee
Expelee
expelee
expelee_official
expelee-co

expelee

Building the Futuristic Blockchain Ecosystem

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantess against the sale of team tokens or the removal of liquidity by the project audited in this document.

Always do your own research and project yourselves from being scammed. The Expelee team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools.

Under no circumstances did Expelee receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Alway do your own research and protect yourselves from scams.

This document should not be presented as a reason to buy or not buy any particular token. The Expelee team disclaims any liability for the resulting losses.