expelee

Building the Futuristic Blockchain Ecosystem

SECURITY AUDIT REPORT

Lottery

TOKEN OVERVIEW

Risk Findings

Severity	Found
High	4
Medium	2
Low	0
Informational	0

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	**16 October 2023**

CONTRACT DETAILS

Contract Address: 0xEaf884ca7c53f2fB541daA0caf66025e112A06F3

Name: Lottery

Decimals: 18

Network: Ethereum

Contract Type: Lottery

Owner: 0x8c0093Ae2705e98D65122C18404f0F469388b05f

Deployer: 0xB91e327B776BCDa3D7931E4221744F928F796c78

Checksum:
2718c4053d4524c4bddcb10ab310739072f73cd9

Testnet version:
The tests conducted were performed on the contract deployed on a private chain (forge foundry)

AUDIT METHODOLOGY

Audit Details

Our comprehensive audit report provides a full overview of the audited system's architecture, smart contract codebase, and details on any vulnerabilities found within the system.

Audit Goals

The audit goal is to ensure that the project is built to protect investors and users, preventing potentially catastrophic vulnerabilities after launch , that lead to scams and rugpulls.

Code Quality

Our analysis includes both automatic tests and manual code analysis for the following aspects:

- Exploits
- Back-doors
- Vulnerability
- Accuracy
- Readability

Tools

- DE
- Open Zeppelin
- Code Analyzer
- Solidity Code
- Compiler
- Hardhat

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

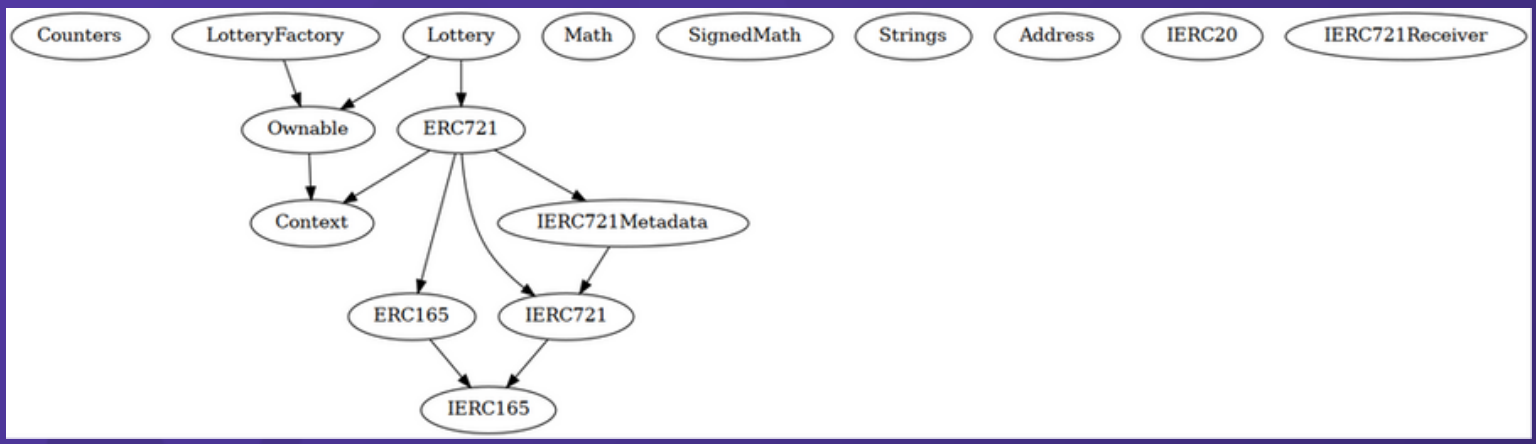
Issues on this level are minor details and warning that can remain unfixed.

Informational

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

expelee

INHERITANCE TREES



FUNCTION DETAILS

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
Counters	Library			
└	current	Internal 🔒		
└	increment	Internal 🔒	🛑	
└	decrement	Internal 🔒	🛑	
└	reset	Internal 🔒	🛑	
IERC165	Interface			
└	supportsInterface	External ❗		NO ❗
ERC165	Implementation	IERC165		
└	supportsInterface	Public ❗		NO ❗
Math	Library			
└	max	Internal 🔒		
└	min	Internal 🔒		
└	average	Internal 🔒		
└	ceilDiv	Internal 🔒		
└	mulDiv	Internal 🔒		
└	mulDiv	Internal 🔒		
└	sqrt	Internal 🔒		
└	sqrt	Internal 🔒		
└	log2	Internal 🔒		
└	log2	Internal 🔒		
└	log10	Internal 🔒		
└	log10	Internal 🔒		
└	log256	Internal 🔒		
└	log256	Internal 🔒		
SignedMath	Library			
└	max	Internal 🔒		

expelee

FUNCTION DETAILS

└	min	Internal 🔒		
└	average	Internal 🔒		
└	abs	Internal 🔒		
Strings	Library			
└	toString	Internal 🔒		
└	toString	Internal 🔒		
└	toHexString	Internal 🔒		
└	toHexString	Internal 🔒		
└	toHexString	Internal 🔒		
└	equal	Internal 🔒		
Context	Implementation			
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
Ownable	Implementation	Context		
└	<Constructor>	Public ❗	🛑	NO ❗
└	owner	Public ❗		NO ❗
└	_checkOwner	Internal 🔒		
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
└	_transferOwnership	Internal 🔒	🛑	
Address	Library			
└	isContract	Internal 🔒		
└	sendValue	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionStaticCall	Internal 🔒		
└	functionStaticCall	Internal 🔒		
└	functionDelegateCall	Internal 🔒	🛑	
└	functionDelegateCall	Internal 🔒	🛑	
└	verifyCallResultFromTarget	Internal 🔒		
└	verifyCallResult	Internal 🔒		
└	_revert	Private 🔐		
IERC20	Interface			
└	totalSupply	External ❗		NO ❗
└	balanceOf	External ❗		NO ❗
└	transfer	External ❗	🛑	NO ❗
└	allowance	External ❗		NO ❗
└	approve	External ❗	🛑	NO ❗
└	transferFrom	External ❗	🛑	NO ❗

expelee

FUNCTION DETAILS

LotteryFactory	Implementation	Ownable		
└	<Constructor>	Public ❗	🛑	NO ❗
└	whitelistAddress	External ❗	🛑	onlyOwner
└	whitelistAddresses	External ❗	🛑	onlyOwner
└	setAffiliateFeeOnCreate	External ❗	🛑	onlyOwner
└	setPlatformAddress	External ❗	🛑	onlyOwner
└	setApiAddress	External ❗	🛑	onlyOwner
└	setPlatformFee	External ❗	🛑	onlyOwner
└	setCreationFee	External ❗	🛑	onlyOwner
└	withdrawEther	External ❗	🛑	onlyOwner
└	withdrawtoken	External ❗	🛑	onlyOwner
└	createLottery	Public ❗	💵	NO ❗
└	validatePrizeDistribution	Internal 🔒		
IERC721	Interface	IERC165		
└	balanceOf	External ❗		NO ❗
└	ownerOf	External ❗		NO ❗
└	safeTransferFrom	External ❗	🛑	NO ❗
└	safeTransferFrom	External ❗	🛑	NO ❗
└	transferFrom	External ❗	🛑	NO ❗
└	approve	External ❗	🛑	NO ❗
└	setApprovalForAll	External ❗	🛑	NO ❗
└	getApproved	External ❗		NO ❗
└	isApprovedForAll	External ❗		NO ❗
IERC721Metadata	Interface	IERC721		
└	name	External ❗		NO ❗
└	symbol	External ❗		NO ❗
└	tokenURI	External ❗		NO ❗
IERC721Receiver	Interface			
└	onERC721Received	External ❗	🛑	NO ❗
ERC721	Implementation	Context, ERC165, IERC721, IERC721Metadata		
└	<Constructor>	Public ❗	🛑	NO ❗
└	supportsInterface	Public ❗		NO ❗
└	balanceOf	Public ❗		NO ❗
└	ownerOf	Public ❗		NO ❗
└	name	Public ❗		NO ❗
└	symbol	Public ❗		NO ❗
└	tokenURI	Public ❗		NO ❗
└	_baseURI	Internal 🔒		
└	approve	Public ❗	🛑	NO ❗
└	getApproved	Public ❗		NO ❗
└	setApprovalForAll	Public ❗	🛑	NO ❗
└	isApprovedForAll	Public ❗		NO ❗

FUNCTION DETAILS

└	transferFrom	Public ❗	🛑	NO❗
└	safeTransferFrom	Public ❗	🛑	NO❗
└	safeTransferFrom	Public ❗	🛑	NO❗
└	_safeTransfer	Internal 🔒	🛑	
└	_ownerOf	Internal 🔒		
└	_exists	Internal 🔒		
└	_isApprovedOrOwner	Internal 🔒		
└	_safeMint	Internal 🔒	🛑	
└	_safeMint	Internal 🔒	🛑	
└	_mint	Internal 🔒	🛑	
└	_burn	Internal 🔒	🛑	
└	_transfer	Internal 🔒	🛑	
└	_approve	Internal 🔒	🛑	
└	_setApprovalForAll	Internal 🔒	🛑	
└	_requireMinted	Internal 🔒		
└	_checkOnERC721Received	Private 🔐	🛑	
└	_beforeTokenTransfer	Internal 🔒	🛑	
└	_afterTokenTransfer	Internal 🔒	🛑	
└	__unsafe_increaseBalance	Internal 🔒	🛑	
Lottery	Implementation	ERC721, Ownable		
└	<Constructor>	Public ❗	🛑	ERC721
└	purchaseLottery	External ❗	🛑	NO❗
└	getPurchasedTickets	External ❗		NO❗
└	setWinners	External ❗	🛑	NO❗
└	whitelistAddresses	External ❗	🛑	onlyOwner

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

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			

expelee

HIGH RISK FINDING

Category: Loss of funds
Subject: Whitelisted wallets are able to withdraw fee ETH from the contract
Status: Open
Severity: High
Impact:
When a wallet is whitelisted, it can withdraw fee ETH from the contract when creating a lottery.

Proof of concept:
- Creation fee of a pool is 1 ether
- affiliationFeeOnCreation is 10 (10%)
- Owner whitelists Alice wallet
- Alice creates a lottery contract and receives 0.1 ether in return, because alice doesn't have to send any values (paying fee), but receives a portion of creation fee anyways

```
//@audit a whitelisted wallet is able to withdraw all ETH from the contract
function createLottery(
    string memory _name,
    uint _ticketPrice,
    uint _maxTickets,
    uint _endDate,
    LotteryFee memory _feeParams,
    uint _maxTicketsPerWallet,
    uint[] memory _prizeDistribution
) public payable {
    //@AUDIT whitelisted wallet doesn't have to pay any ETH
    if (!isWhitelisted[msg.sender]) {
        require(msg.value == creationFee, "Incorrect fee");
    }

    uint affiliateCommission = (creationFee * affiliateFeeOnCreation) / 100;

    if (_feeParams.affiliateWallet != address(0)) {
        //@AUDIT : affiliateCommission (a portion of creationFee) is sent to affiliateWallet which is an arbitrary address
        //chosed by msg.sender
        _feeParams.affiliateWallet.transfer(affiliateCommission);
    }
```

Mitigation:
Do not send **affiliateCommision** to **affiliateWallet** if wallet is whitelisted.

HIGH RISK FINDING

Category: Loss of funds
Subject: Whitelisted wallets are able to withdraw fee ETH from the contract
Status: Open
Severity: High
Impact:
When a wallet is whitelisted, it can withdraw fee ETH from the contract when creating a lottery.

Proof of concept:
- Creation fee of a pool is 1 ether
- affiliationFeeOnCreation is 10 (10%)
- Owner whitelists Alice wallet
- Alice creates a lottery contract and receives 0.1 ether in return, because alice doesn't have to send any values (paying fee), but receives a portion of creation fee anyways

```
//@audit a whitelisted wallet is able to withdraw all ETH from the contract
function createLottery(
    string memory _name,
    uint _ticketPrice,
    uint _maxTickets,
    uint _endDate,
    LotteryFee memory _feeParams,
    uint _maxTicketsPerWallet,
    uint[] memory _prizeDistribution
) public payable {
    //@AUDIT whitelisted wallet doesn't have to pay any ETH
    if (!isWhitelisted[msg.sender]) {
        require(msg.value == creationFee, "Incorrect fee");
    }

    uint affiliateCommission = (creationFee * affiliateFeeOnCreation) / 100;

    if (_feeParams.affiliateWallet != address(0)) {
        //@AUDIT : affiliateCommission (a portion of creationFee) is sent to affiliateWallet which is an arbitrary address
        //chosed by msg.sender
        _feeParams.affiliateWallet.transfer(affiliateCommission);
    }
```

Mitigation:
Do not send **affiliateCommision** to **affiliateWallet** if wallet is whitelisted.

HIGH RISK FINDING

Category: Centralization
Subject: Malicious owner is able to set any arbitrary participate as winner
Status: Open
Severity: High
Impact:
Wining system of the lottery is not based on luck (a random number), in fact owner is able to set any wallet as the winner. Winner will be receiving all the funds in the lottery.

Proof of concept:
as you can see in the below section of the code, api wallet is able to set any wallet(s) as the winner.

```
function setWinners(uint[] memory tokenIds) external {
    //@AUDIT : api wallet can set any wallets as winner
    require(msg.sender == apiWallet, "Not permitted");
    uint _totalFunds = IERC20(feeToken).balanceOf(address(this));
```

Mitigation:

Make sure that winners are determined by a random number.

HIGH RISK FINDING

Category: Loss of funds
Subject: Lottery can not be finished due to lack of allowance if ticketPrice is zero
Status: Open
Severity: High

Impact:
Since contract doesn't have approval on behalf of it self to spend certain tokens, setWinners function will be reverted due to lack of allowance (if ticketPrice == 0)

Proof of concept:
as you can see in the below section of the code, since contact doesn't have allowance to spend its own tokens, setWinners function will always be reverted.

```
if (ticketPrice == 0) {
    uint platformCut = (_totalFunds * platformFee) / 100;
    //@audit since IERC20(feeToken).allowance(address(this), address(this)) == 0
    //below line reverts
    IERC20(feeToken).transferFrom(
        address(this),
        feeAddress,
        platformCut
    );
    _totalFunds = _totalFunds - platformCut;
}
```

Mitigation:
approve contract to spend its own tokens, add this line to **setWinners** function:
_IERC20(feeTOken).approve(address(this), ~uint256(0));

MEDİUM RISK FINDING

Category: Validation
Subject: Max tickets can be bypassed
Status: Open
Severity: Medium
Impact:
In Lottery contract, maxTickets can be bypassed, because function is incorrectly checking if this value is reached or not

Proof of concept:
as you can see in the below section of the code, amount is required to be less than maxtickets, however this condition is not enough to make sure that maxTickets is not exceeded.
Example:
- MaxTickets is 100
- Alice and 9 of her friends purchase 10 tickets each (max ticket per wallet = 10)
- Bob can still purchase 10 more tickets because the condition only requires “10” to be less than “100”

```
function purchaseLottery(uint amount, address _affiliateWallet) external {
    require(isRunning, "Lottery is over");
    require(amount > 0, "Amount must be greater than 0");
    //@AUDIT : condition requires amount to be less than maxTickets
    //but is not checking whether maxTickets are already bought or not
    require(amount <= maxTickets, "Max tickets exceeded");
    require(
        myTickets[msg.sender] + amount <= maxTicketsPerWallet,
        "Max tickets per wallet exceeded"
    );
```

Mitigation:
Accumulate purchased tickets and check whether this value is reached or not.

MEDIUM RISK FINDING

Category: Validation
Subject: endDate is not checked
Status: Open
Severity: Medium
Impact:
setWinners in lottery contract is not checking whether end date is reached or not

Proof of concept:
No checks at setWinners to validate block.timestamp to be more than endDate

```
function setWinners(uint[] memory tokenIds) external {
    require(msg.sender == apiWallet, "Not permitted");
    uint _totalFunds = IERC20(feeToken).balanceOf(address(this));
```

Mitigation:
Check whether block.timestamp is greater than endDate
require(block.timestamp > endDate);

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.