expelee

Building the Futuristic Blockchain Ecosystem

# Audit Report FOR

# Braavos Water Dance

# 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 |
| KYC Verification | Not Done |
| Audit Date | 28 Sep 2022 |

-Team Expelee

# PROJECT DESCRIPTION

## Braavos-Water-Dance

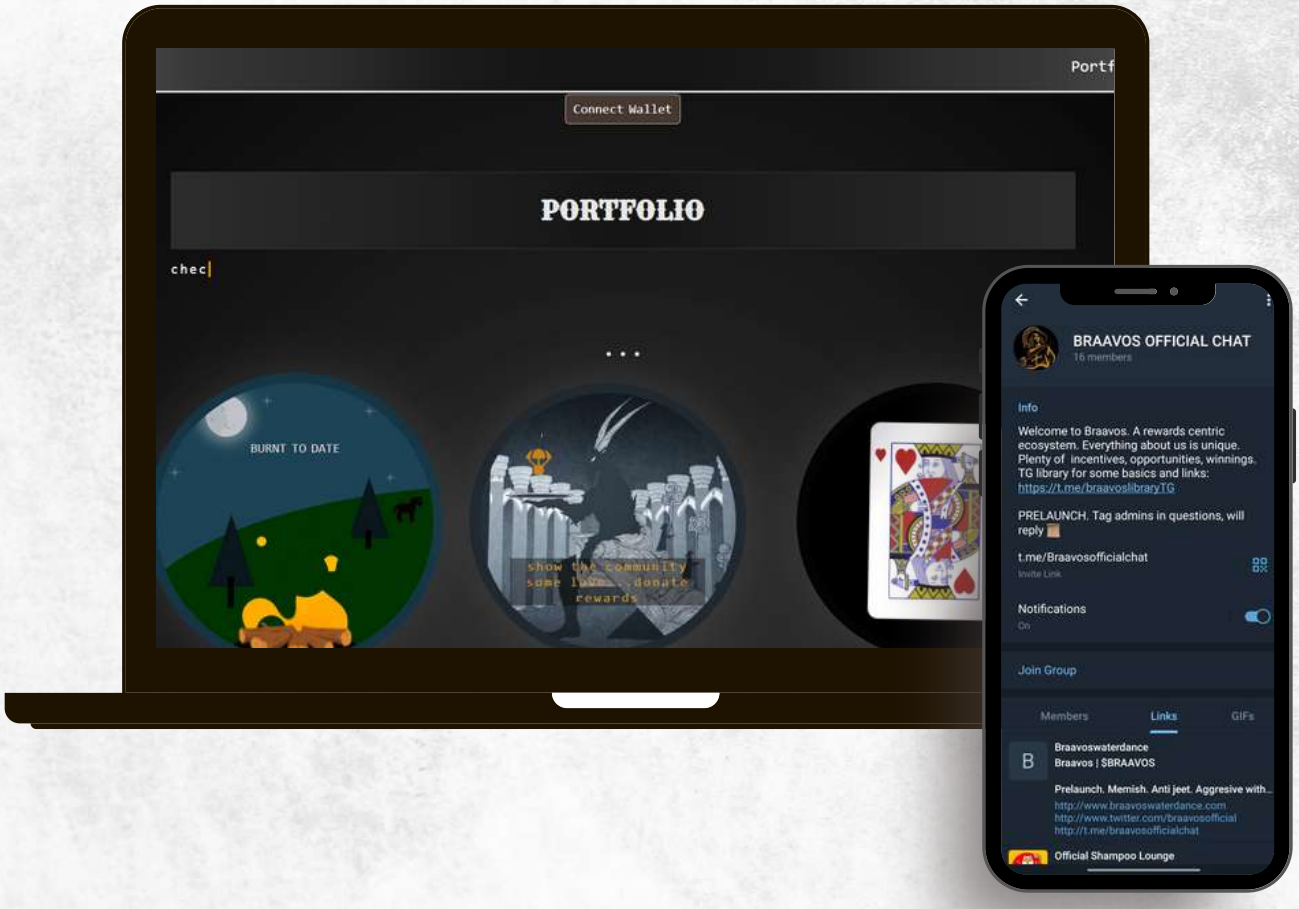
Braavos brings circular economic model to the passive income sector.
First fully reward centric ecosystem, we amplify the utility of rewards.
First tax reimbusing project (a challenge awaits).
First to allow wallets to configure taxes. Attempt the temple challenge.

expelee

expelee.com

# Social Media Profiles

## Bravoos Water Dance




https://www.braavoswaterdance.com/

https://t.me/braavosofficialchat

https://twitter.com/braavosofficial

**It's always good to check the social profiles of the project, before making your investment.**

**-Team Expelee**

# CONTRACT DETAILS

Token Name
**Braavos-Water-Dance**

Symbol
**BRAAVOS**

Network
**BSC**

Language
**Solidity**

Contract Address (Verified)
**0xadaF57a3918818B57ad32A80F30Ba9466f7D6aa1**

Token Type
**ERC 20**

Decimals
**18**

Compiler
**v0.8.4+commit.c7e474f2**

License
**MIT license**

Contract SHA-256 Checksum:
**398c9538d9b7dd3f5d12ce6120d20705c2321c59fef62433de571e93004564e0**

# 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
- Complier
- Hardhat

# VULNERABILITY CHECKLIST

| | |
|---|---|
| Design Logic | Passed |
| Compiler warnings. | Passed |
| Private user data leaks | Passed |
| Timestamp 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 Zeppelin module | Passed |
| Fallback function security | Passed |

# RISK CLASSIFICATION

When performing smart contract audits, our specialists look for known vulnerabilities as well as logical and access 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

Information level is to offer suggestions for improvement of efficacy or security for features with a risk free factor.

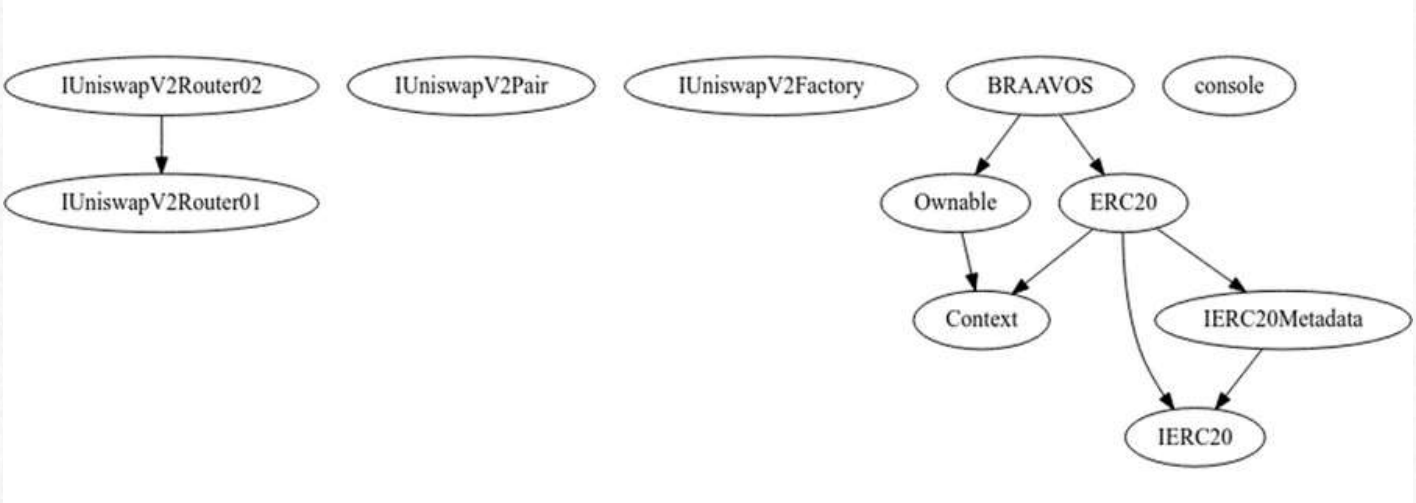
# AUDIT SUMMARY

## Ownership:

0x9ebb4c9e810ceb88422303c96541a04117262235, owner privileges are discussed in findings report

## Contracts & Inheritance Tree:

all of below contractsare in this audit scope



# Summary


- a multi-task ERC20 token, you can stake your tokens, play russian roulette, have a change to win a portion of total accumulated rewards and create auctions and leases, basically an ecosystem with many functionalities that you can use to earn a profit.
- there could be up to 50% tax on buys and 25% on sells
- owner is able to set a max amount for buying selling and holding amount
- you can stake your tokens and get rewards
- you can play russian roulette 4 times a day and win a percentage of total rewards
- on every buy there is a change for you to win a percentage of total accumulated rewards (In BNB), this reward can be very big based on the volume
- you can lease your tokens to other people in exchange for receiving a certain amount of eth, if they feel that your lease rates are reasonable, they can lease your tokens and get a reward based on them
- you can stake your tokens and get rewards based on your staked amount
- you can invite your friends to this ecosystem, they can use your address as a referall code to join the ecosystem and this will be an advantage for you to receive more rewards
- you can add eth liquidity to the pool and get 20% more eth in return if there is enough eth in the contract

# MANUAL AUDIT

## Severity Criteria

Expelee assesses the severity of disclosed vulnerabilities according to a methodology based on
OWASP standards.
Vulnerabilities are divided into three primary risk categories: high, medium, and low.

High-level considerations for vulnerabilities span the followingkey 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 | | | |

# Findings Summary

- High Risk Findings:18 (RESOLVED)
- Medium Risk Findings:6 (RESOLVED)
- Low Risk Findings:4
- Suggestions & discussion: 5
- Gas Optimizations : 4

# High Risk Findings

Centralization – Owner is able to mint any amount of tokens using addLiquidity function
this is a public function which is only callable by owner, owner is able to mint any amount of tokens.
total supply of the token is 0 currently, so basically this is the function that owner uses to mint initial supply tokens as well as adding the liquidity. But this function is not a one time use function and can be called later again.
Suggestion:
set reasonable restrictions on this function to not be able to mint unlimited tokens.
Alleviation:
fixed, added: require(_totalSupply < 500000001 * 1e18);

Centralization – Owner is able to drain liquidity pool:
Since all the LP tokens are going to owner's wallet, owner or a hacker that got access to this wallet are able to drain the liquidity pool, same thing is true for fundLiquidity function where normal wallets can add liquidity for the token received but LP tokens will go to owner's wallet.
Alleviation:
intention was to burn LP tokens and renounce ownership soon after launch like most projects to gain trust and good Dextools ratings.

**Centralization – Owner is able to bypass lock time:**
Owner can not sell his tokens before 90 days from launch, but is able to send his tokens to another wallet, this way owner is able to pass lock time

```
 if (sender == owner() && recipient == _pairAddress){
require(block.timestamp > (_tradingStart + (1 * 60 * 60 * 24 * 90)),"team tokens locked");
}
```

**Suggestion:**
disable all kind of transfers for owner's wallet (or the wallet that holds locked tokens)
**Alleviation:**
intention was to burn LP tokens and renounce ownership soon after launch like most projects to gain trust and good Dextools ratings.

**Centralization – owner is able to take dev fund tokens (minted to contract when added liquidity) before starting the trade by using _takeBusDev function, this amount of tokens are suppose to be locked for 180 days after launch**
**Allevation:**
Function has been removed totally.

**Centralization – owner is able to set a limit for transferring/buying/selling amount using setMaxHoldings function, this limit has a minimum of 1, which means that normal wallets are not able to buy / sell or transfer more than this amount**
**Allevation:**
Fixed: require(_maxHoldings <= 10000000 * 1e18);

**Logical - The generated random number is not truly random:**
since blockchain is a deterministic system its hard to produce a truly – random number on-chain, its not easy to predict this current random number but its not impossible to do so. If a hacker find a way to predict the randomHash value then he/she is able to win the games every time.
**Suggestion:**
use chainlink oracles to create a strong random number off-chain
https://docs.chain.link/docs/vrf/v2/examples/get-a-random-number/
**Allevation:**
fixed, require(block.timestamp - _lastWin[msg.sender] > (1 * 60 * 60 * 24),"cooldown 24hrs");
resorted to limiting wins per day to 1 max. As Russian roulette is used to reduce sell taxes only,

Logical - underflow error at _swap function:
_liquidityFundingRepaid will be 0 until someone withdraw his provided liquidity rewards using liquidityWithdraw function. Now if gap is more than 0 and no one used liquidityWithdraw function:

```
if(gap > 0 && swapValue >= gap){
_liquidityRepayPool += gap;
swapValue -= gap;
_liquidityFundingRepaid -= gap;
}
```

this if statement leads to a revert due to underflow error at last line.
Suggestion:
make sure that _liquidityFundingRepaid is greater than or equal to gap

```
if(_liquidityFundingRepaid >= gap){
    _liquidityFundingRepaid -= gap;
}
```

same thing is happening at this line
_liquidityFundingRepaid -= swapValue;
Allevation:
fixed if(gap > 0 && swapValue >= gap && _liquidityFundingRepaid >= gap){
To cover both

Logical – anyone is able to clear donors of another address:
at removeBeneficiary function:
uint index = _privateList[account].myDonorsIndex[msg.sender];
if msg.sender is not in donors of account, index will be equal to 0. and index 0 of the account's donors will be deleted, attacker can repeat this action to delete all the donors of account.
Suggestions:
there is many approaches that could solve this issue, our suggestion is:
at addBeneficiary function, instead of setting:
_privateList[account].myDonors.length – 1;
as index of donor, set:
_privateList[account].myDonors.length
as index of donor.
and then at removeBeneficiary function use donorIndex – 1 to delete donor from the array.
After removing donor from the array set its index to 0:
_privateList[account].myDonorsIndex[msg.sender] = 0
By doing this you can also add this require statement
require(_privateList[account].myDonorsIndex[msg.sender] != 0, "you are not in donors list!")
Allevation:
Fixed As Suggested!

**Logical – Can not add a wallet as bot:**
at addBot function, Owner is able to only add liquidity pool or pancake router as a bot:
if(account != address(_router) || account != _pairAddress){
revert();
}
**Suggestion:**
change the if statement to:
if(account == address(_router) || account == _pairAddress){...}
**Alleviation:**
fixed if(account == address(_router) || account == _pairAddress){revert();}

**Logical – Setting buyTime and total plays to 0 at start of _reimburseBuyTax function:**
_acptMap[account].buytime = 0;
_myplaysTotal[account] = 0;
By doing this "holdtime" variable will be equal to block.timestamp which is a very big time.
By doing this "_myplaysTotal[account]" will be equal to zero and we wont be eligible for the middle condition:
if(holdtime > (1 * 60 * 60 * 24) && _myplaysTotal[account] >= 6){
**Suggestion:**
assign this variable to 0 at end of the function

**Logical – at _reimburseBuyTax function _reimbursePool is reduced even if account is not for any condition:**
if account is taxExcluded and has 4 or 5 total plays he still wont be eligible for any of the conditions
if(!isTaxExcluded(account) && _myplaysTotal[account] < 6){
if(holdtime > (1 * 60 * 60 * 24) && _myplaysTotal[account] >= 6){
}else if(isTaxExcluded(account) && _playCount[account] <= 3){
but _reimbursePool will still be reduced by reimbursedAmt
**Alleviation:**
fixed //covers wallets that didnt play, didnt win and 4 or 5 plays only
if(!isTaxExcluded(account) && _myplaysTotal[account] < 6){
4or5 plays are covered here.

Logical – deducting gap amount from repaid ethereums to liquidty funders
at _swap function gap or swapValue is deducted from _liquidityFundingRepaid.
its not clear why we are doing this since we are only increasing gap after each _swap:
uint256 gap = _liquidityFundingDue - _liquidityFundingRepaid;
Suggestion:
its not clear what is the purpose behind this functionality, but we thing that this might be a mistake since _liquidtyFundingRepaid increases by X every time a liquidity funder withdraw X amount of its profits (due amount)
Alleviation:
fixed, mistake as you spotted it, have since removed both adjustments to _liquidityFundingRepaid

Logical – division rounded to 0 at unstake function
at this line of code:
_lastRewardBasis[msg.sender] += amount/_stakeMap[msg.sender].amount * _totalEthReflectedST[msg.sender];
the first require statements declares that amount must be <= _stakeMap[msg.sender].amount
require(_stakeMap[msg.sender].amount >= amount);
if so then:
amount / _stakeMap[msg.sender].amount
is either 0 or 1, this causes _lastRewardBasis[msg.sender] to be zero
Suggestion:
add a numerator to avoid 0 rounding errors
Alleviation:
fixed: if(_totalEthReflectedST[msg.sender] > 0){
    //if claimed then update reflectionBasis with amount claimed rewards from staked tokens
    _lastRewardBasis[msg.sender] += (amount * _totalEthReflectedST[msg.sender]) /_stakeMap[msg.sender].amount;
  }
Added additional check.
Also added a require() check to make sure amount input is always > 0

Logical – division rounded to 0 at reimbursedRewradCheck
if(_stakeMap[msg.sender].amount >= _reimburseQueue[account]){
return myreimburseRewards;
}else{
uint256 portion = _stakeMap[msg.sender].amount / _reimburseQueue[account];
return portion;
}

if first if statement is not true, then else block means that _stakeMap[msg.sender].amount < _reimbursedQueue[account] and this function will return zero.

Suggestion:

add a numerator to avoid 0 rounding errors

Alleviation:

fixed; uint256 portion = (myreimburseRewards * _stakeMap[msg.sender].amount) / _reimburseQueue[account];

Logical – potential underflow at _takeBuyTax function:

if _reimburseRate is more than _buyRate, then totalFees will be less than reimburseAmnt and this transaction will revert at this point

_feeTokens += (totalFees – reimburseAmnt);

Suggestion:

make sure that reimburseAmnt is greater than totalFees

Alleviation:

Solved as suggested

Logical - totalFees does not include reimburseAmnt:

at _takeBuyTax function, "totalFees" does not include reimburseAmnt (they are calculated seperately) but we are still deducting reimburseAmnt from totalFees:

_feeTokens += (totalFees – reimburseAmnt);

Suggestion:

if this is not intentional, then remove this line

Alleviation:

fixed: _feeTokens += totalFees;

It was a mistake to subtract reimburse again there

Logical - all transactions at _tradingStartBlock will be reverted:

since tx.origin is not equal to _pairAddress or _router, addBot(tx.origin) will revert the transactions

Alleviation:

fixed; //anti snipe the mev mechants

```
if(sender == _pairAddress){
 _tradeBlock[recipient] = block.number;//when you buy we log block
}
if(recipient == _pairAddress && _tradeBlock[sender] == block.number){
  _addBot(recipient);
}
```

Might not work to catch all bots if they transfer first

Logical – a lessor can be disabled from concluding its lease:
require(_sharesLeaseAmnt[msg.sender] == _shareMap[msg.sender].amount);
its possible for _sharesLeaseAmnt[msg.sender] to be more than _shareMap[msg.sender].amount, if such thing happen, msg.sender will not be able to conclude its lease ever.
Suggestion:
change the require statement to
require(_sharesLeaseAmnt[msg.sender] >= _shareMap[msg.sender].amount);
Alleviation:
fixed: require(_sharesLeaseAmnt[msg.sender] >= _shareMap[msg.sender].amount);
Also added the below require to to ensure wallets can create 1 lease at a time & not change _sharesLeaseAmnt[msg.sender]
function createShareLease(uint256 tokens, uint256 ethRequired, uint256 lease_days) external {
require(_sharesLeaseAmnt[msg.sender] == 0);//create 1 lease at a time

# Medium Risk Findings

Centralization – owner is able to set buy taxes and sell taxes up to 25% and also reimburseRate up to 25%, this means 50% tax on buys and 25% on sell (75% in total) if owner set all of them to maximum.
Alleviation:
Fixed: Max buy or sell tax in setting is 25 not 50. Please notice that under the function _getBuyTax(uint256 amount)
tokens are only taxed buyTax, the send amount does not have any other amount removed. Then in the
notereimburse
variable is only passed back and used to note down how much to reimburse latter during a sell if wallet qualifies.

Logical – at addBeneficiary function, someone can add himself to donors of address unlimited times.
_privateList[account].myDonors.push(msg.sender);
uint index = _privateList[account].myDonors.length - 1;
Suggestion:
if this functionality is not intentional add this require statement:
require(!_claimBeneficiary[msg.sender] != account, “Already added as Donor”);
Alleviation:
fixed: require(!_claimBeneficiary[msg.sender] != account, "Already added as Donor");
Added as suggested

Logical – redundant or invalid condition at _transfer function:
tx.origin is equal to address of holder whether its buying or selling, so it will never be equal to _pairAddress
if (tx.origin == address(_pairAddress)) {
Suggestion:
you can delete this condition, even if tx.origin is equal to _pairAddress this condition is still most likely redundant
Alleviation:
-Removed

Logical – someone is able to push its address to _lessorsArray unlimited times:
by passing 0 for all arguments in :
function createShareLease(uint256 tokens, uint256 ethRequired, uint256 lease_days)
a hacker can add its wallet to _lessorsArray each time, since there is no restirctions
_lessorsArray.push(msg.sender);
Suggestion:
set proper restrictions and conditions, so that a wallet can not be added to _lessorsArray multiple times
Alleviation:
Fixed: added
require(_sharesLeaseAmnt[msg.sender] == 0);//create 1 lease at a time
in previous fix of unlimited leases

- Since everyone is able to start an auction, a hacker can just use auctionStart() function and start an unnecessary auction. If this happens, no one is able to start a new auction before auction time ends (max 24 hours).
Alleviation:
Fixed: added modifier admin() to make sure only admin can start auctions when needed.

- Setting _tradingStartBlock to wrong number is irreversible and we have to redeploy the contract

# Low Risk Findings

Centralization – owner is able to set _poachSwitch to false and hence disable claiming reflections
Alleviation:
Fixed: poachswitch is used to disable rewards in order to have them consumed in buying competitions (lucky dip) or to have them auctioned.

Logical – adding duplicate holders:
At addLiquidity function, while we are minting to the contract, number of holders will still increase (at _mint function) even that we are sending funds to the same wallet.
_mint(address(this), initialpooltokens);
_mint(address(this), _liquidityReserves);
_mint(owner(), funding);
_mint(address(this), funding);
Suggestion:
set _holders initial value to 2 (since we are only minting to contract and owner address) and delete:
_holders += 1
from _mint function.
Alleviation:
Fixed: added a condition:

```
if(balanceOf(account) == 0){
    _holders += 1;
```

Logical – using transfer function to transfer ETH:
Do not use transfer function for transferring ETH, since EIP-1884 gas cost of some opcodes changed, for example SLOAD gas cost increases from 400 to 800
Suggestion:
use low level calls for transferring ETH and also make sure to set a reentrancy guard on all the functions that are sending eth to an arbitrary address
Alleviation:
fixed: now using send , within a dedicated _paymentETH function

```
function _paymentETH(address receiver, uint256 amount) internal {
   (bool sent, ) = receiver.call{value: amount}("");
   require(sent, "Failed to send Ether");
 }
```

Logical – at _playRussianRoulette, number of players increases even for same players:
_players += 1; (Resolved ✓)
Alleviation:
fixed: removed _players variable totally from contract

## Gas Optimizations

- console library never used inside the contract, delete this library to reduce contract size a lot
- - redundant check in _transfer fuction : ...recipient == address(_router));, recipient will not be equal to recipient in any of trades (Resolved ✅)
- - this require statement here is redundant:
  require(auctions[_auctionID].currentbidder != msg.sender,"wait");
  if someone bids for the auction, he/she is not able to bid again because of this line
  _lastAuction[msg.sender] = _auctionID;(Resolved ✅)

## Suggestions

- you can use up to 3 indexed arguments in events, make sure to use this 3 in your events
- Solidity versions >=0.8.0 include checked arithmetic operations and underflow/overflow (for signedandunsigned integers) by default, making the usage of multiple SafeMath libraries redundant. The underflow/overflow check is performedat every step in a calculation.

# 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 guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document.

Always Do your own research and protect 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. Always 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.