SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

BNBWOLF
[BNBWOLF]
BEP20

0x502DD4564E4F78581c57154902A903c793D318f5

Table of Contents

Global Disclaimer

This document serves as a disclaimer for the crypto smart contract audit conducted by Skeleton Ecosystem. The purpose of the audit was to review the codebase of the smart contracts for potential vulnerabilities and issues. It is important to note the following:

Limited Scope: The audit is based on the code and information available up to the audit completion date. It does not cover external factors, system interactions, or changes made after the audit. The audit itself can not guarantee 100% safaty and can not detect common scam methods like farming and developer sell-out.

No Guarantee of Security: While we have taken reasonable steps to identify vulnerabilities, it is impossible to guarantee the complete absence of security risks or issues. The audit report provides an assessment of the contract's security as of the audit date.

Continued Development: Smart contracts and blockchain technology are evolving fields. Updates, forks, or changes to the contract post-audit may introduce new risks that were not present during the audit.

Third-party Code: If the smart contract relies on third-party libraries or code, those components were not thoroughly audited unless explicitly stated. Security of these dependencies is the responsibility of their respective developers.

Non-Exhaustive Testing: The audit involved automated analysis, manual review, and testing under controlled conditions. It is possible that certain vulnerabilities or issues may not have been identified.

Risk Evaluation: The audit report includes a risk assessment for identified vulnerabilities. It is recommended that the development team carefully reviews and addresses these risks to mitigate potential exploits.

Not Financial Advice: This audit report is not intended as financial or investment advice. Decisions regarding the use, deployment, or investment in the smart contract should be made based on a comprehensive assessment of the associated risks.

By accessing and using this audit report, you acknowledge and agree to the limitations outlined above. Skeleton Ecosystem and its auditors shall not be held liable for any direct or indirect damages resulting from the use of the audit report or the smart contract itself.

Please consult with legal, technical, and financial professionals before making any decisions related to the smart contract.

BNBWOLF BEP20

Overview

Contract Name	BNBWolf
Ticker/Simbol	BNBWolf
Blockchain	Binance Smart Chain Bep20
Contract Address	0x502DD4564E4F78581c57154902A903c793D318f5
Creator Address	0xD77481b2e4Ab100f6a456909EBdD126A0C131890
Current Owner Address	0xD77481b2e4Ab100f6a456909EBdD126A0C131890
Contract Explorer	https://bscscan.com/token/0x502dd4564e4f78581c57154902a903c793d318f5
Compiler Version	v0.8.17+commit.8df45f5f
License	None
Optimisation	Yes with 200 Runs
Total Supply	500,000,000,000,000 BNBWolf
Decimals	18

Creation/Audit

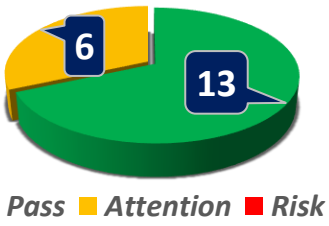
Contract Deployed	3 Aug 2023
Audit Created	06-Sept-23 09:00 UTC
Audit Update	V 0.1

Verified Socials

Website	https://www.bnbwolf.org/
Telegram	https://t.me/BNBWolfbsc
Twitter	https://www.twitter.com/BNBWolfbsc

BNBWOLF BEP20

Contract Function Analysis

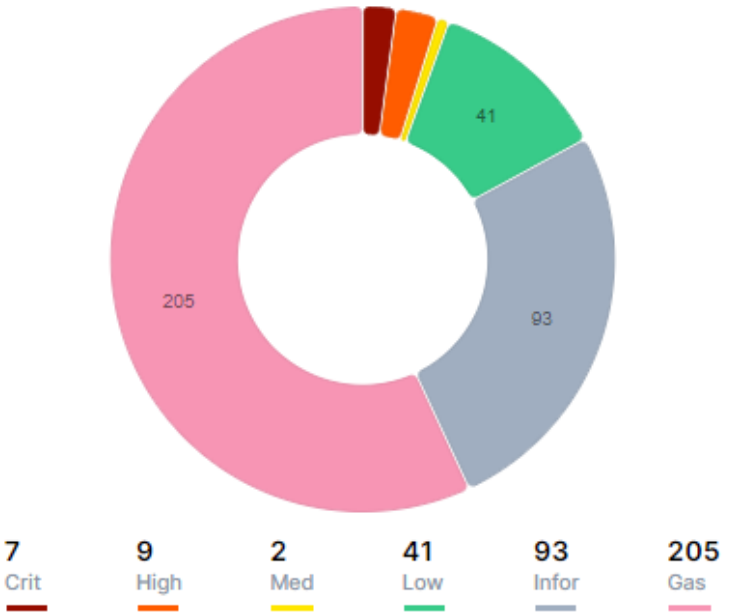


Pass Attention Item Risky Item

Contract Verified	✓	The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Renounce	⚠	Current Owner: 0xD77481b2e4Ab100f6a456909EBdD126A0C131890 Attention marked functions can be modified and used.
Buy Tax	**10 %** (max 10%)	Shows the taxes for purchase transactions. Above 10% may be considered a high tax rate. More than 50% tax rate means may not be tradable. **Setfee max found: 10%**
Sell Tax	**10 %** (max 10%)	Shows the taxes for sell transactions. Above 10% may be considered a high tax rate. More than 50% tax rate means may not be tradable. **Setfee max found: 10%**
Honeypot Analyse	✓	Holder is able to buy and sell. If honeypot: The contract blocks sell transfer from holder wallet. Multiple events may cause honeypot. Trading disabled, extremely high tax
Liqudity Status	⚠	Locked on 05.08.2023: 80% Unicrypt for 3620 days. Note! Initial liqudity tokens scanned. For new LP Lockers allways re-check with skeleton scanner on telegram.
Trading Disable Functions	✓	No trading suspendable function found. If a suspendable code is included, the token maybe neither be bought or sold (honeypot risk). If contract is renounced this function can't be used. ⚠ If there is authorised hidden owner, or there is Retrieve Ownership Function, the trading disable function may be used!
Set Fees function	⚠ (max 10%)	Fee Setting function found. **Setfee max found: 10%**
Proxy Contract	✓	The proxy contract means contract owner can modifiy the function of the token and possibly effect the price. The Owner is not the creator but the creator may have authorisation to change functions.
Mint Function	✓	No mint function found. Mint function is transparent or non-existent. Hidden mint functions may increase the amount of tokens in circulation and effect the price of the token. Owner can mint new tokens and sell. If contract is renounced this function can't be used.

Balance Modifier Function		No Balance Modifier function found. If there is a function for this, the contract owner can have the authority to modify the balance of tokens at other addresses. For example revoke the bought tokens from the holders wallet. Common form of scam: You buy the token, but it's disappearing from your wallet. ⚠ If contract is renounced this function still can be used as auto self Destruct
Whitelist Function		Whitelist Function Found. If there is a function for this Developer can set zero fee or no max wallet size for adresses (for example team wallets can trade without fee. Can cause farming) If there is a whitelist, some addresses may not be able to trade normally (honeypot risk
Hidden Owner Analysis		No authorised hidden owner found. For contract with a hidden owner, developer can still manipulate the contract even if the ownership has been abandoned. Fake renounce.
Retrieve Ownership Function		No functions found which can retrieve ownership of the contract. If this function exists, it is possible for the project owner to regain ownership even after relinquishing it. Also known as fake renounce.
Self Destruct Function		No Self Destruct function found. If this function exists and is triggered, the contract will be destroyed, all functions will be unavailable, and all related assets will be erased.
Specific Tax Changing Function		Specific Tax Changing Functions found. If it exists, the contract owner may set a very outrageous tax rate for assigned address to block it from trading. Can assign all wallets at once!
Trading Cooldown Function		Trading Cooldown Function found. If there is a trading cooldown function, the user will not be able to sell the token within a certain time or block after buying. Like a temporary honeypot.
Max Transaction and Holding Modify Function		No Max Transaction and Holding Modify function found. If there is a function for this, the maximum trading amount or maximum position can be modified. Can cause honeypot
Transaction Limiting Function		No Transaction Limiter Function Found. The number of overall token transactions may be limited (honeypot risk)

Contract Safety and Weakness



INCORRECT ACCESS CONTROL	7
USE OF TX.ORIGIN	2
UNCHECKED TRANSFER	1
UNCHECKED ARRAY LENGTH	1
APPROVE FRONT-RUNNING ATTACK	5
DEPRECATED SAFEAPPROVE	1
DIVISION BY ZERO	1
USE OWNABLE2STEP	1
LONG NUMBER LITERALS	1
USE OF FLOATING PRAGMA	1

OUTDATED COMPILER VERSION	1
MISSING EVENTS	35
FUNCTION RETURNS TYPE AND NO RE...	2
RETURN INSIDE LOOP	1
MISSING PAYABLE IN CALL FUNCTION	3
MISSING STATE VARIABLE VISIBILITY	8
MISSING INDEXED KEYWORDS IN EVE...	1
REQUIRE WITH EMPTY MESSAGE	11
UNUSED RECEIVE FALLBACK	1
IN-LINE ASSEMBLY DETECTED	2

MISSING UNDERSCORE IN NAMING VA...	55
USE CALL INSTEAD OF TRANSFER OR ...	4
BLOCK VALUES AS A PROXY FOR TIME	7
USE OF SAFEMATH LIBRARY	2
CODE OPTIMIZATION BY USING MAX ...	1
FUNCTION SHOULD BE EXTERNAL	11
UNNECESSARY CHECKED ARITHMETI...	2
SPLITTING REQUIRE STATEMENTS	19
GAS OPTIMIZATION IN INCREMENTS	2
CUSTOM ERRORS TO SAVE GAS	1

FUNCTION SHOULD RETURN STRUCT	1
UNNECESSARY DEFAULT VALUE INITI...	1
ARRAY LENGTH CACHING	2
CHEAPER INEQUALITIES IN IF()	10
VARIABLES DECLARED BUT NEVER US...	4
CHEAPER INEQUALITIES IN REQUIRE()	30
CONSTANT STATE VARIABLES	1
CHEAPER CONDITIONAL OPERATORS	9
OPTIMIZING ADDRESS ID MAPPING	12
STORAGE VARIABLE CACHING IN MEM...	84

⚠ Incorrect Access Control (7 item)

```
1373        if (_isExcluded[account]) return _tOwned[account];
1374        return tokenFromReflection(_rOwned[account]);
1375    }
1376
1377    function transfer(address recipient, uint256 amount)
1378        public
1379        override
1380        returns (bool)
1381    {
1382        _transfer(_msgSender(), recipient, amount);
1383        return true;
1384    }
1385
1386    function allowance(address owner, address spender)
```

Function	Severity	Remedation
Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract. The contract ProToken is importing an access control library @openzeppelin/contracts/access/Ownable.sol but the function **transfer** is missing the modifier onlyOwner.	Severity : Critical	It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

```
1394
1395    function approve(address spender, uint256 amount)
1396        public
1397        override
1398        returns (bool)
1399    {
1400        _approve(_msgSender(), spender, amount);
1401        return true;
1402    }
1403
```

Function	Severity	Remedation
Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract. The contract ProToken is importing an access control library @openzeppelin/contracts/access/Ownable.sol but the function **approve** is missing the modifier onlyOwner.	Severity : Critical	It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

```
1401        return true;
1402    }
1403
1404    function transferFrom(
1405        address sender,
1406        address recipient,
1407        uint256 amount
1408    ) public override returns (bool) {
1409        _transfer(sender, recipient, amount);
1410        _approve(
1411            sender,
1412            _msgSender(),
1413            _allowances[sender][_msgSender()].sub(
1414                amount,
1415                "ERC20: transfer amount exceeds allowance"
1416            )
1417        );
1418        return true;
1419    }
1420
1421    function increaseAllowance(address spender, uint256 addedValue)
1422        public
```

Function	Severity	Remedation
Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract. The contract ProToken is importing an access control library @openzeppelin/contracts/access/Ownable.sol but the function **transferFrom** is missing the modifier onlyOwner.	Severity : Critical	It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

```
1452    }
1453
1454    function deliver(uint256 tAmount) public {
1455        address sender = _msgSender();
1456        require(
1457            !_isExcluded[sender],
1458            "Excluded addresses cannot call this function"
1459        );
1460        (uint256 rAmount, , , , , ) = _getValues(tAmount);
1461        _rOwned[sender] = _rOwned[sender].sub(rAmount);
1462        _rTotal = _rTotal.sub(rAmount);
1463        _tFeeTotal = _tFeeTotal.add(tAmount);
1464    }
1465
1466    function reflectionFromToken(uint256 tAmount, bool deductTransferFee)
1467        public
1468        view
1469        returns (uint256)
1470    {
```

Function	Severity	Remedation
Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract. The contract ProToken is importing an access control library @openzeppelin/contracts/access/Ownable.sol but **the function deliver is missing the modifier onlyOwner.**	Severity : Critical	It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

```
2161
2162      function process(uint256 gas)
2163          public
2164          returns (
2165              uint256,
2166              uint256,
2167              uint256
2168          )
2169      {
2170          uint256 numberOfTokenHolders = tokenHoldersMap.keys.length;
2171
2172          if (numberOfTokenHolders == 0) {
2173              return (0, 0, lastProcessedIndex);
2174          }
2175
2176          uint256 _lastProcessedIndex = lastProcessedIndex;
2177
2178          uint256 gasUsed = 0;
2179
2180          uint256 gasLeft = gasleft();
2181
2186              _lastProcessedIndex++;
2187
2188              if (_lastProcessedIndex >= tokenHoldersMap.keys.length) {
2189                  _lastProcessedIndex = 0;
2190              }
2191
2192              address account = tokenHoldersMap.keys[_lastProcessedIndex];
2193
2194              if (canAutoClaim(lastClaimTimes[account])) {
2195                  if (processAccount(payable(account), true)) {
2196                      claims++;
2197                  }
2198              }
2199              iterations++;
2200              uint256 newGasLeft = gasleft();
2201              if (gasLeft > newGasLeft) {
2202                  gasUsed = gasUsed.add(gasLeft.sub(newGasLeft));
2203              }
2204              gasLeft = newGasLeft;
2205          }
2206
2207          lastProcessedIndex = _lastProcessedIndex;
2208          return (iterations, claims, lastProcessedIndex);
2209      }
```

Function	Severity	Remedation
Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract. The contract ProToken is importing an access	Severity : Critical	It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

control library @openzeppelin/contracts/access/Ownable.sol but **the function process is missing the modifier onlyOwner.**		

```
2364
2365        function processDividendTracker(uint256 gas) external {
2366            (uint256 iterations, uint256 claims, uint256 _lastProcessedIndex) = process(
2367                gas
2368            );
2369            emit ProcessedDividendTracker(
2370                iterations,
2371                claims,
2372                _lastProcessedIndex,
2373                false,
2374                gas,
2375                tx.origin
2376            );
2377        }
2378
2379        function claim() external {
2380            processAccount(payable(msg.sender), false);
2381        }
```

Function	Severity	Remedation
Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract. The contract ProToken is importing an access control library @openzeppelin/contracts/access/Ownable.sol **but the function processDividendTracker is missing the modifier onlyOwner.**	Severity : Critical	It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

```
2375              tx.origin
2376          );
2377      }
2378
2379      function claim() external {
2380          processAccount(payable(msg.sender), false);
2381      }
2382
2383          /* Dividend management functions*/
2384      function distributeDividends(uint256 amount) internal {
2385          require(_tDividendTotal > 0);
```

Function	Severity	Remedation
Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract. The contract ProToken is importing an access control library @openzeppelin/contracts/access/Ownable.sol but **the function claim is missing the modifier onlyOwner.**	Severity : Critical	It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

⚠ Use of TX origin (2 item)

```
1785                    _lastProcessedIndex,
1786                    true,
1787                    gas,
1788                    tx.origin
1789                );
2372                _lastProcessedIndex,
2373                false,
2374                gas,
2375                tx.origin
2376            );
2377        }
2378
2379        function claim() external {
2380            processAccount(payable(msg.sender), false);
```

Function	Severity	Remedation
In Solidity, tx.origin is a global variable that returns the address of the account that sent the transaction. Using the variable for authorization could make a contract vulnerable. For example, if an authorized account calls a malicious contract which triggers it to call the vulnerable contract that passes an authorization check since tx.origin returns the original sender of the transaction which in this case is the authorized account.	Severity : High	tx.origin should not be used for authorization in smart contracts. It does have some legitimate use cases, for example, To prevent external contracts from calling the current contract, you can implement a require of the form require(tx.origin == msg.sender). This prevents intermediate contracts from calling the current contract, thus limiting the contract to regular codeless addresses.

⚠ Unchecked Transfer (1 Item)

```
619     ) internal {
620         _callOptionalReturn(
621             token,
622             abi.encodeWithSelector(token.transferFrom.selector, from, to, value)
623         );
624     }
625
```

Function	Severity	Remedation
Some tokens do not revert the transaction when the transfer or transferFrom fails and returns False. Hence we must check the return value after calling the transfer or transferFrom function.	Severity : High	Use OpenZeppelin SafeERC20's safetransfer and safetransferFrom functions.

⚠ Unchecked Array Lenght (1 Item)

```
2343
2344      function includeInReward(address account) external onlyOwner {
2345          require(_isExcluded[account], "Already excluded");
2346          for (uint256 i = 0; i < _excluded.length; i++) {
2347              if ( excluded[i] == account) {
2348                  _excluded[i] = _excluded[_excluded.length - 1];
```

Function	Severity	Remedation
Ethereum is a very resource-constrained environment. Prices per computational step are orders of magnitude higher than with centralized providers. Moreover, Ethereum miners impose a limit on the total number of Gas consumed in a block. If array.length is large enough, the function exceeds the block gas limit, and transactions calling it will never be confirmed. for (uint256 i = 0; i < array.length ; i++) { cosltyFunc(); } This becomes a security issue if an external actor influences array.length. E.g., if an array enumerates all registered addresses, an adversary can register many addresses, causing the problem described above.	Severity : High	Either explicitly or just due to normal operation, the number of iterations in a loop can grow beyond the block gas limit, which can cause the complete contract to be stalled at a certain point. Therefore, loops with a bigger or unknown number of steps should always be avoided.

⚠ Approve Frontrunning Attack (5 Item)

```
1392        return _allowances[owner][spender];
1393    }
1394
1395    function approve(address spender, uint256 amount)
1396        public
1397        override
1398        returns (bool)
1399    {
1400        _approve(_msgSender(), spender, amount);
1401        return true;
1402    }
1403
1404    function transferFrom(
```

Function	Severity	Remedation
The **approve()** method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account. Meanwhile, if the sender decides to change the amount and sends another approve transaction, the receiver can notice this transaction before it's mined and can extract tokens from both the transactions, therefore, ending up with tokens from both the transactions. This is a front-running attack affecting the ERC20 Approve function. The function approve can be front-run by abusing **the _approve** function.	Severity : High	Only use the approve function of the ERC/BEP standard to change the allowed amount to 0 or from 0 (wait till transaction is mined and approved). Token owner just needs to make sure that the first transaction actually changed allowance from N to 0, i.e., that the spender didn't manage to transfer some of N allowed tokens before the first transaction was mined. Such checking is possible using advanced blockchain explorers such as [Etherscan.io](https://etherscan.io/) Another way to mitigate the threat is to approve token transfers only to smart contracts with verified source code that does not contain logic for performing attacks like described above, and to accounts owned by the people you may trust.

```
1401        return true;
1402    }
1403
1404    function transferFrom(
1405        address sender,
1406        address recipient,
1407        uint256 amount
1408    ) public override returns (bool) {
1409        _transfer(sender, recipient, amount);
1410        _approve(
1411            sender,
1412            _msgSender(),
1413            _allowances[sender][_msgSender()].sub(
1414                amount,
1415                "ERC20: transfer amount exceeds allowance"
1416            )
1417        );
1418        return true;
1419    }
1420
1421    function increaseAllowance(address spender, uint256 addedValue)
```

Function	Severity	Remedation
The **transferFrom()** method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account. Meanwhile, if the sender decides to change the amount and sends another approve transaction, the receiver can notice this transaction before it's mined and can extract tokens from both the transactions, therefore, ending up with tokens from both the transactions. This is a front-running attack affecting the ERC20 Approve function.	Severity : High	Only use the approve function of the ERC/BEP standard to change the allowed amount to 0 or from 0 (wait till transaction is mined and approved). Token owner just needs to make sure that the first transaction actually changed allowance from N to 0, i.e., that the spender didn't manage to transfer some of N allowed tokens before the first transaction was mined. Such checking is possible using advanced blockchain explorers such as [Etherscan.io](https://etherscan.io/) Another way to mitigate the threat is to approve token transfers only to smart contracts with verified source code that does not contain logic for performing attacks like described above, and to accounts owned by the people you may trust.

The function transferFrom can be front-run by abusing the _approve function.		

```
1886            emit SwapAndLiquify(half, newBalance, otherHalf);
1887        }
1888    }
1889
1890    function swapTokensForBNB(uint256 tokenAmount) private {
1891        // generate the uniswap pair path of token -> weth
1892        address[] memory path = new address[](2);
1893        path[0] = address(this);
1894        path[1] = pcsV2Router.WETH();
1895
1896        _approve(address(this), address(pcsV2Router), tokenAmount);
1897
1898        // make the swap
1899        pcsV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
1900            tokenAmount,
1901            0, // accept any amount of ETH
1902            path,
1903            address(this),
1904            block.timestamp
1905        );
1906    }
1907
1908    function swapBNBForTokens(uint256 amount) private {
```

Function	Severity	Remedation
The **swapTokensForBNB**() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account.	Severity : High	Only use the approve function of the ERC/BEP standard to change the allowed amount to 0 or from 0 (wait till transaction is mined and approved). Token owner just needs to make sure that the first transaction actually changed allowance from N to 0, i.e., that the spender didn't manage to transfer some of N allowed tokens before the first transaction was mined. Such checking is possible using advanced blockchain explorers

Meanwhile, if the sender decides to change the amount and sends another approve transaction, the receiver can notice this transaction before it's mined and can extract tokens from both the transactions, therefore, ending up with tokens from both the transactions. This is a front-running attack affecting the ERC20 Approve function. The function swapTokensForBNB can be front-run by abusing the _approve function.		such as [Etherscan.io](https://etherscan.io/) Another way to mitigate the threat is to approve token transfers only to smart contracts with verified source code that does not contain logic for performing attacks like described above, and to accounts owned by the people you may trust.

Function	Severity	Remedation
The **swapTokensForFeeToken()** method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account. Meanwhile, if the sender decides to change the amount and sends another approve transaction, the receiver can notice this transaction before it's mined and can extract tokens from both the transactions, therefore, ending up with tokens from both the transactions.	Severity : High	Only use the approve function of the ERC/BEP standard to change the allowed amount to 0 or from 0 (wait till transaction is mined and approved). Token owner just needs to make sure that the first transaction actually changed allowance from N to 0, i.e., that the spender didn't manage to transfer some of N allowed tokens before the first transaction was mined. Such checking is possible using advanced blockchain explorers such as [Etherscan.io](https://etherscan.io/) Another way to mitigate the threat is to approve token transfers only to smart contracts with verified source code that does not contain logic for performing attacks like described above, and to

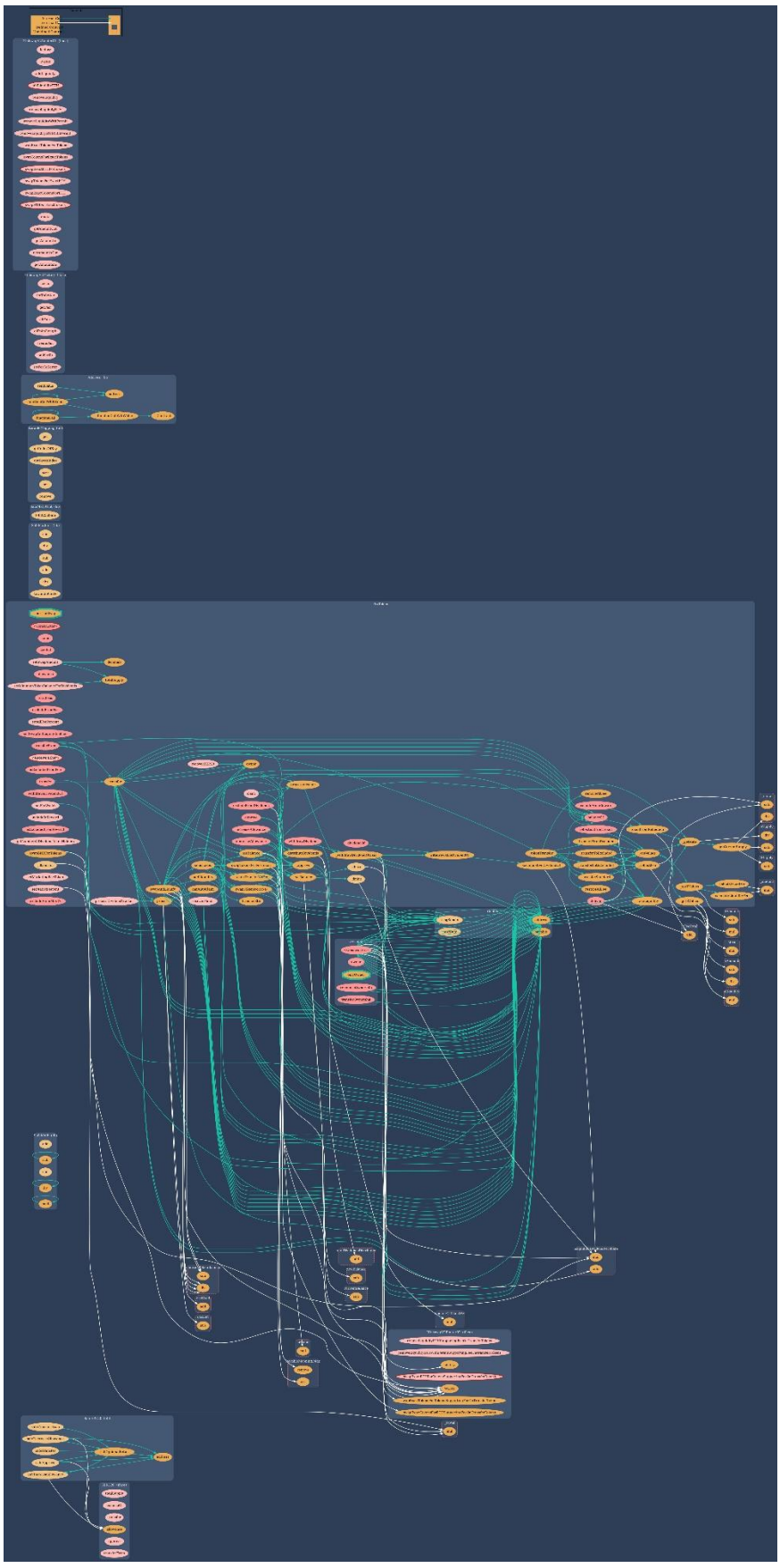
This is a front-running attack affecting the ERC20 Approve function. The function swapTokensForFeeToken can be front-run by abusing the _approve function.		accounts owned by the people you may trust.

```
1957            IERC20(feeToken).transfer(receiver, newBalance);
1958        }
1959    }
1960
1961    function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
1962        // approve token transfer to cover all possible scenarios
1963        _approve(address(this), address(pcsV2Router), tokenAmount);
1964
1965        address liquidAddr = dead;
1966
1967        if (!burnAutomaticGeneratedLiquidity) {
1968            liquidAddr = owner();
1969        }
1970        // add the liquidity
1971        pcsV2Router.addLiquidityETH{value: ethAmount}(
1972            address(this),
1973            tokenAmount,
1974            0, // slippage is unavoidable
1975            0, // slippage is unavoidable
1976            liquidAddr,
1977            block.timestamp
1978        );
1979    }
1980
1981    //this method is responsible for taking all fee, if takeFee is true
```

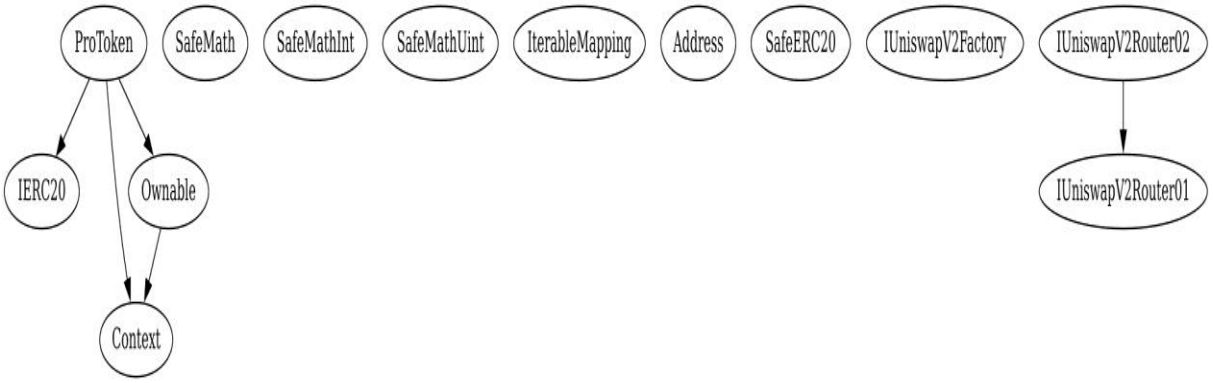
Function	Severity	Remedation
The **addLiquidity**() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account.	Severity : High	Only use the approve function of the ERC/BEP standard to change the allowed amount to 0 or from 0 (wait till transaction is mined and approved). Token owner just needs to make sure that the first transaction actually changed allowance from N to 0, i.e., that the spender didn't manage to transfer some of N allowed tokens before the first transaction was mined. Such checking is possible using advanced blockchain explorers

Meanwhile, if the sender decides to change the amount and sends another approve transaction, the receiver can notice this transaction before it's mined and can extract tokens from both the transactions, therefore, ending up with tokens from both the transactions. This is a front-running attack affecting the ERC20 Approve function. The function addLiquidity can be front-run by abusing the _approve function.		such as [Etherscan.io](https://etherscan.io/) Another way to mitigate the threat is to approve token transfers only to smart contracts with verified source code that does not contain logic for performing attacks like described above, and to accounts owned by the people you may trust.

Contract Flow Graph



Inheritance Graph



Contract Descriptions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External ❗		NO❗
	balanceOf	External ❗		NO❗
	transfer	External ❗	🛑	NO❗
	allowance	External ❗		NO❗
	approve	External ❗	🛑	NO❗
	transferFrom	External ❗	🛑	NO❗
SafeMath	Library			
	add	Internal 🔒		
	sub	Internal 🔒		
	sub	Internal 🔒		
	mul	Internal 🔒		
	div	Internal 🔒		
	div	Internal 🔒		
	mod	Internal 🔒		
	mod	Internal 🔒		

Context	Implementation			
	_msgSender	Internal 🔒		
	_msgData	Internal 🔒		
SafeMathInt	Library			
	mul	Internal 🔒		
	div	Internal 🔒		
	sub	Internal 🔒		
	add	Internal 🔒		
	abs	Internal 🔒		
	toUint256Safe	Internal 🔒		
SafeMathUint	Library			
	toInt256Safe	Internal 🔒		
IterableMapping	Library			
	get	Internal 🔒		
	getIndexOfKey	Internal 🔒		
	getKeyAtIndex	Internal 🔒		
	size	Internal 🔒		
	set	Internal 🔒	🛑	

	remove	Internal 🔒	🛑	
Address	Library			
	isContract	Internal 🔒		
	sendValue	Internal 🔒	🛑	
	functionCall	Internal 🔒	🛑	
	functionCall	Internal 🔒	🛑	
	functionCallWithValue	Internal 🔒	🛑	
	functionCallWithValue	Internal 🔒	🛑	
	_functionCallWithValue	Private 🔐	🛑	
SafeERC20	Library			
	safeTransfer	Internal 🔒	🛑	
	safeTransferFrom	Internal 🔒	🛑	
	safeApprove	Internal 🔒	🛑	
	safeIncreaseAllowance	Internal 🔒	🛑	
	safeDecreaseAllowance	Internal 🔒	🛑	
	_callOptionalReturn	Private 🔐	🛑	

Ownable	Implementation	Context		
		Public ❗	🛑	NO❗
	owner	Public ❗		NO❗
	renounceOwnership	Public ❗	🛑	onlyOwner
	transferOwnership	Public ❗	🛑	onlyOwner
IUniswapV2Factory	Interface			
	feeTo	External ❗		NO❗
	feeToSetter	External ❗		NO❗
	getPair	External ❗		NO❗
	allPairs	External ❗		NO❗
	allPairsLength	External ❗		NO❗
	createPair	External ❗	🛑	NO❗
	setFeeTo	External ❗	🛑	NO❗
	setFeeToSetter	External ❗	🛑	NO❗
IUniswapV2Router01	Interface			
	factory	External ❗		NO❗
	WETH	External ❗		NO❗
	addLiquidity	External ❗	🛑	NO❗

	addLiquidityETH	External ❗	💵	NO❗
	removeLiquidity	External ❗	🛑	NO❗
	removeLiquidityETH	External ❗	🛑	NO❗
	removeLiquidityWithPermit	External ❗	🛑	NO❗
	removeLiquidityETHWithPermit	External ❗	🛑	NO❗
	swapExactTokensForTokens	External ❗	🛑	NO❗
	swapTokensForExactTokens	External ❗	🛑	NO❗
	swapExactETHForTokens	External ❗	💵	NO❗
	swapTokensForExactETH	External ❗	🛑	NO❗
	swapExactTokensForETH	External ❗	🛑	NO❗
	swapETHForExactTokens	External ❗	💵	NO❗
	quote	External ❗		NO❗
	getAmountOut	External ❗		NO❗

	getAmountIn	External ❗		NO❗
	getAmountsOut	External ❗		NO❗
	getAmountsIn	External ❗		NO❗
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💵	NO❗

	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
ProToken	Implementation	Context, IERC20, Ownable		
		Public ❗	💵	NO❗
	name	Public ❗		NO❗
	symbol	Public ❗		NO❗
	decimals	Public ❗		NO❗
	totalSupply	Public ❗		NO❗
	balanceOf	Public ❗		NO❗
	transfer	Public ❗	🛑	NO❗
	allowance	Public ❗		NO❗
	approve	Public ❗	🛑	NO❗
	transferFrom	Public ❗	🛑	NO❗
	increaseAllowance	Public ❗	🛑	NO❗
	decreaseAllowance	Public ❗	🛑	NO❗
	totalFees	Public ❗		NO❗
	deliver	Public ❗	🛑	NO❗

	reflectionFromToken	Public ❗		NO❗
	tokenFromReflection	Public ❗		NO❗
	excludeFromFee	Public ❗	🛑	onlyOwner
	setAllFeePercent	External ❗	🛑	onlyOwner
	setSwapAndLiquifyEnabled	Public ❗	🛑	onlyOwner
	setSwapAmount	External ❗	🛑	onlyOwner
		External ❗	💵	NO❗
	_reflectFee	Private 🔐	🛑	
	_getValues	Private 🔐		
	_getTValues	Private 🔐		
	_getRValues	Private 🔐		
	_getRate	Private 🔐		
	_getCurrentSupply	Private 🔐		
	_takeLiquidity	Private 🔐	🛑	
	calculateTaxFee	Private 🔐		
	calculateLiquidityFee	Private 🔐		
	removeAllFee	Private 🔐	🛑	
	restoreAllFee	Private 🔐	🛑	

	isExcludedFromFee	Public ❗		NO❗
	_approve	Private 🔒	🛑	
	_transfer	Private 🔒	🛑	
	swapAndLiquify	Private 🔒	🛑	lockTheSwap
	swapTokensForBNB	Private 🔒	🛑	
	swapBNBForTokens	Private 🔒	🛑	
	swapTokensForFeeToken	Private 🔒	🛑	
	addLiquidity	Private 🔒	🛑	
	_tokenTransfer	Private 🔒	🛑	
	_transferStandard	Private 🔒	🛑	
	_transferToExcluded	Private 🔒	🛑	
	_transferFromExcluded	Private 🔒	🛑	
	_transferBothExcluded	Private 🔒	🛑	
	_tokenTransferNoFee	Private 🔒	🛑	
	transferEth	Private 🔒	🛑	
	recoverFunds	External ❗	🛑	onlyOwner

	recoverBEP20	External ❗	🛑	onlyOwner
	sendTaxes	Internal 🔒	🛑	
	process	Public ❗	🛑	NO❗
	processAccount	Internal 🔒	🛑	
	excludeFromDividends	Public ❗	🛑	onlyOwner
	canAutoClaim	Private 🔐		
	dividendOf	Public ❗		NO❗
	withdrawableDividendOf	Public ❗		NO❗
	withdrawnDividendOf	Public ❗		NO❗
	accumulativeDividendOf	Public ❗		NO❗
	_withdrawDividendOfUser	Internal 🔒	🛑	
	withdrawDividend	Public ❗	🛑	NO❗
	setMinimumTokenBalanceForDividends	External ❗	🛑	onlyOwner
	excludeFromReward	Public ❗	🛑	onlyOwner

	includeInReward	External ❗	🛑	onlyOwner
	isExcludedFromReward	Public ❗		NO❗
	getNumberOfDividendTokenHolders	External ❗		NO❗
	processDividendTracker	External ❗	🛑	NO❗
	claim	External ❗	🛑	NO❗
	distributeDividends	Internal 🔒	🛑	
	_dtransfer	Internal 🔒	🛑	
	_dmint	Internal 🔒	🛑	
	_dburn	Internal 🔒	🛑	
	_setBalance	Internal 🔒	🛑	
	setBalance	Private 🤏	🛑	
	setFeeWallet	External ❗	🛑	onlyOwner
	setMarketingFeeToken	External ❗	🛑	onlyOwner
	setMaxTxPercent	External ❗	🛑	onlyOwner
	excludeFromMaxTx	Public ❗	🛑	onlyOwner

 Function can modify state

 Function is payable

File Name SHA-1 Hash

c:\Solidity\bnbwolf.sol abbec6e14db97c724aa36af5b95a27a63b474079

Audit Scope

Audit Method.

Our smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. Goal: discover errors, issues and security vulnaribilities in the code. Findings getting reported and improvements getting suggested.

Automatic and Manual Review
We are using automated tools to scan functions and weeknesses of the contract. Transfers, integer over-undeflow checks such as all CWE events.

Tools we use:
Visual Studio Code
CWE
SWC
Solidity Scan
SVD

In manual code review our auditor looking at source code and performing line by line examination. This method helps to clarify developer's coding decisions and business logic.

Skeleton Ecosystem

https://skeletonecosystem.com

https://github.com/SkeletonEcosystem/Audits