SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

VAULTGUARD [VAULTG]

BEP 20

0xF249dbA246C9Ac0499CbC32b89fA3C7EC7c7a1f0

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.

Overview

Contract Name	VaultGuard
Ticker/Simbol	VaultG
Blockchain	Binance Smart Chain Bep20
Contract Address	0xF249dbA246C9Ac0499CbC32b89fA3C7EC7c7a1f0
Creator Address	0xff3166faf00EB93CFfF54Fa9a41709365e203913
Current Owner Address	0xff3166faf00EB93CFfF54Fa9a41709365e203913
Contract Explorer	https://bscscan.com/token/0xf249dba246c9ac0499cbc32b89fa3c7ec7c7a1f0
Compiler Version	v0.8.19+commit.7dd6d404 (For token contract)
License	None
Optimisation	Yes with 200 Runs
Total Supply	200,000,000 VaultG
Decimals	18

Creation/Audit

Contract Deployed	Aug-27-2023
Audit Created	31-Aug-23 21:00:00 UTC
Audit Update	V 0.1

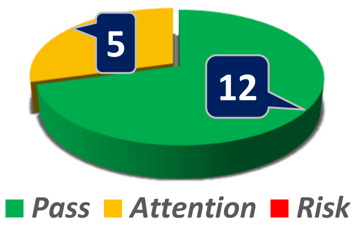
Verified Socials

Website	https://vaultguard.io
Telegram	https://t.me/VaultGuard
Twitter	https://twitter.com/VaultGuardFlash

VAULTGUARD BEP20

Contract Function Analysis

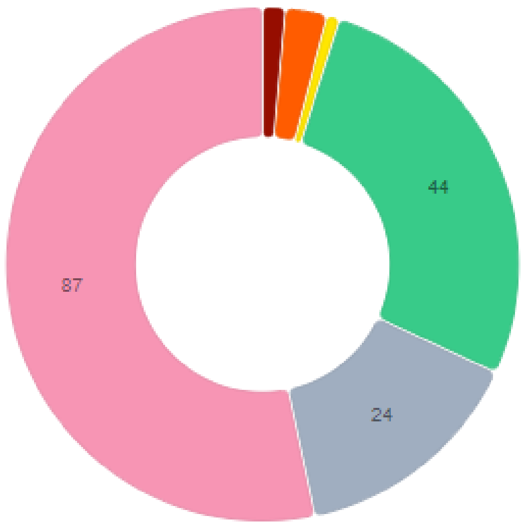
✓ Pass ⚠ Attention Item ⚠ Risky Item



Contract Verified	✓	The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Renounce	⚠	Current Owner: 0xff3166faf00EB93CFfF54Fa9a41709365e203913 Attention marked functions can be modified and used.
Buy Tax	**9 %** (max 25%)	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:** 25% require(totalFees[0] <= 2500 && totalFees[1] <= 2500 && totalFees[2] <= 2500, "TaxesDefaultRouter: Cannot exceed max total fee of 25%");
Sell Tax	**11 %** (max 25%)	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:** **25% require(totalFees[0] <= 2500 && totalFees[1] <= 2500 && totalFees[2] <= 2500, "TaxesDefaultRouter: Cannot exceed max total fee of 25%");**
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 31.08.2023: 94.05% Pinklock for 36523 days. 2.93% Pinklock for 364 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 25%)	Fee Setting function found. **Setfee max found: 25%: require(totalFees[0] <= 2500 && totalFees[1] <= 2500 && totalFees[2] <= 2500, "TaxesDefaultRouter: Cannot exceed max total fee of 25%");**
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	✅	No 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	⚠	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

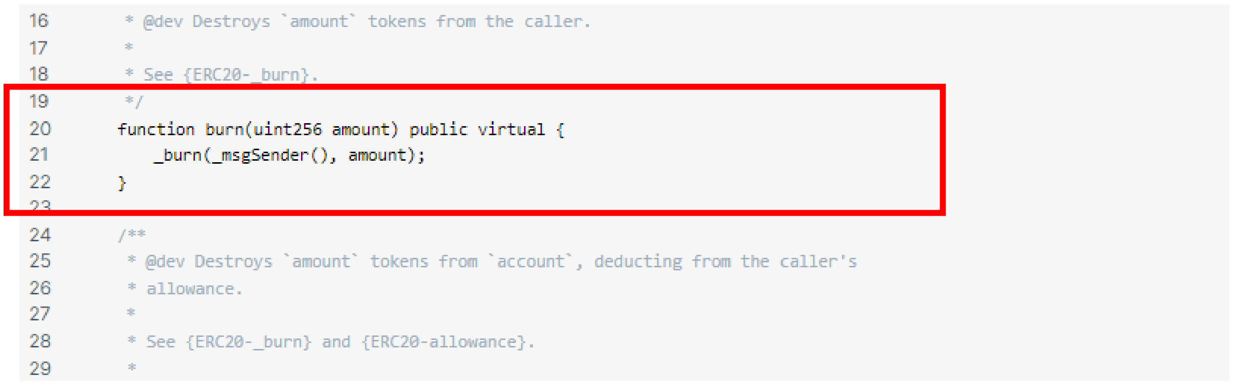


2	4	1	44	24	87
Crit	High	Med	Low	Infor	Gas

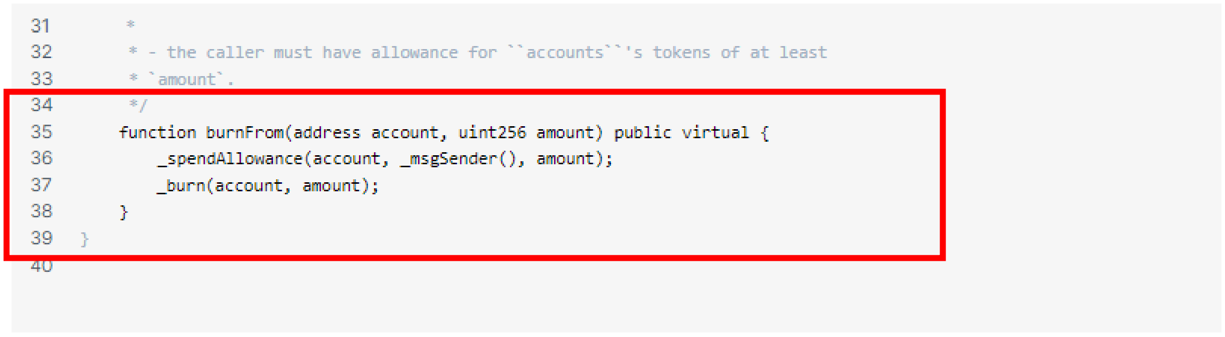
● PUBLIC BURN	2
● UNCHECKED TRANSFER	2
● APPROVE FRONT-RUNNING ATTACK	2
● MODIFIER SIDE EFFECTS	1
● USE OF MULTIPLE PRAGMA VERSIONS	3
● MISCONFIGURED BEFORETOKENTRAN...	4
● COMPILER VERSION TOO RECENT	1
● USE OWNABLE2STEP	2
● MISSING EVENTS	7
● LONG NUMBER LITERALS	3

● OUTDATED COMPILER VERSION	12
● USE OF FLOATING PRAGMA	12
● NAME MAPPING PARAMETERS	4
● BLOCK VALUES AS A PROXY FOR TIME	6
● UNUSED RECEIVE FALLBACK	1
● HARD-CODED ADDRESS DETECTED	5
● MISSING INDEXED KEYWORDS IN EVE...	8
● MULTIPLICATION/DIVISION BY TWO S...	1
● INTERNAL FUNCTIONS NEVER USED	1
● DEFINE CONSTRUCTOR AS PAYABLE	2
● FUNCTION SHOULD BE EXTERNAL	6
● CHEAPER INEQUALITIES IN IF()	2
● SPLITTING REQUIRE STATEMENTS	3
● GAS OPTIMIZATION FOR STATE VARIA...	1
● UNUSED IMPORTS	2
● CHEAPER INEQUALITIES IN REQUIRE()	12
● CHEAPER CONDITIONAL OPERATORS	10
● LONG REQUIRE/REVERT STRINGS	19
● STORAGE VARIABLE CACHING IN MEM...	28

⚠ Public Burn (2 item)



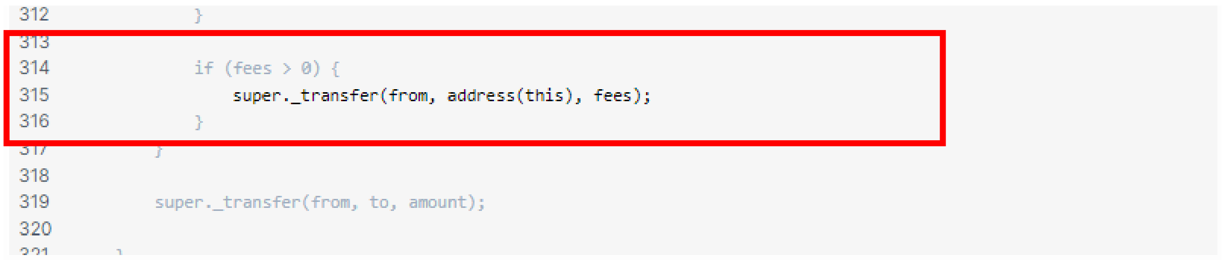
```
     * @dev Destroys `amount` tokens from the caller.
     *
     * See {ERC20-_burn}.
     */
    function burn(uint256 amount) public virtual {
        _burn(_msgSender(), amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, deducting from the caller's
     * allowance.
     *
     * See {ERC20-_burn} and {ERC20-allowance}.
     *
```



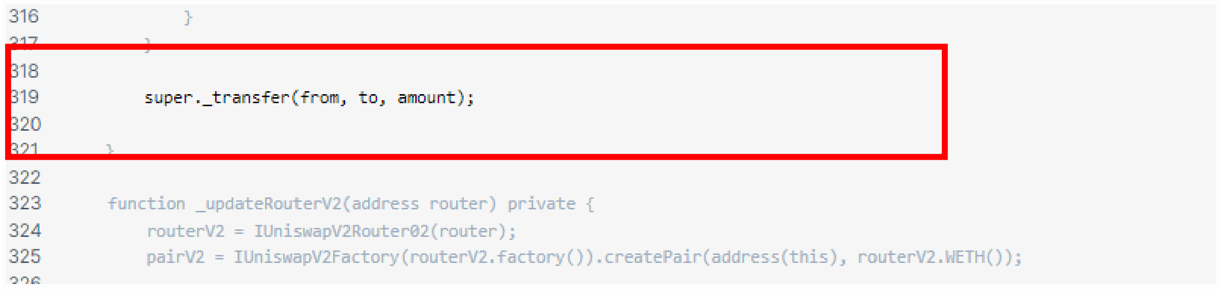
```
     *
     * - the caller must have allowance for ``accounts``'s tokens of at least
     * `amount`.
     */
    function burnFrom(address account, uint256 amount) public virtual {
        _spendAllowance(account, _msgSender(), amount);
        _burn(account, amount);
    }
}
```

Function	Severity	Remedation
The contract was found to be using public or an external burn function. The function was missing access control to prevent another user from burning their tokens. Also, the burn function was found to be using a different address than msg.sender.	Severity : Critical	Consider adding access control modifiers to the burn function to prevent another user from burning their tokens. The burn function should use msg.sender in the _from argument.

⚠ Unchecked Transfer (2 item)



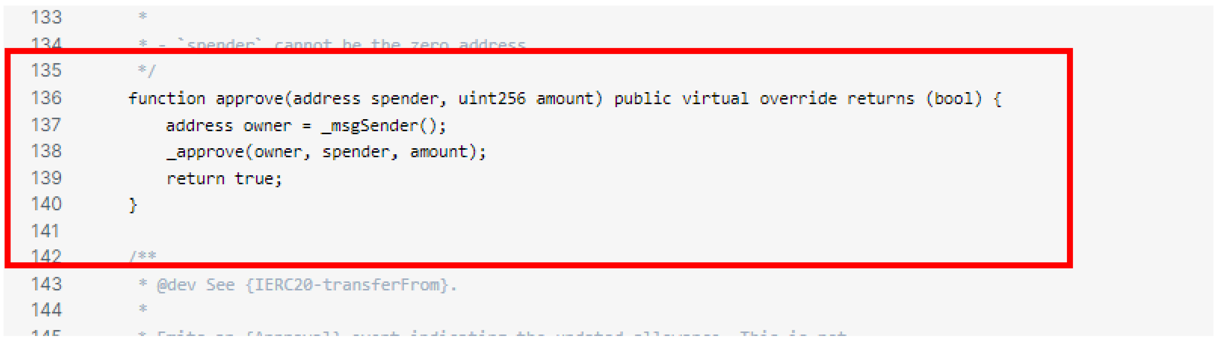
```
            }

            if (fees > 0) {
                super._transfer(from, address(this), fees);
            }
        }

        super._transfer(from, to, amount);

    }
```



```
            }
        }

        super._transfer(from, to, amount);

    }

    function _updateRouterV2(address router) private {
        routerV2 = IUniswapV2Router02(router);
        pairV2 = IUniswapV2Factory(routerV2.factory()).createPair(address(this), routerV2.WETH());
```

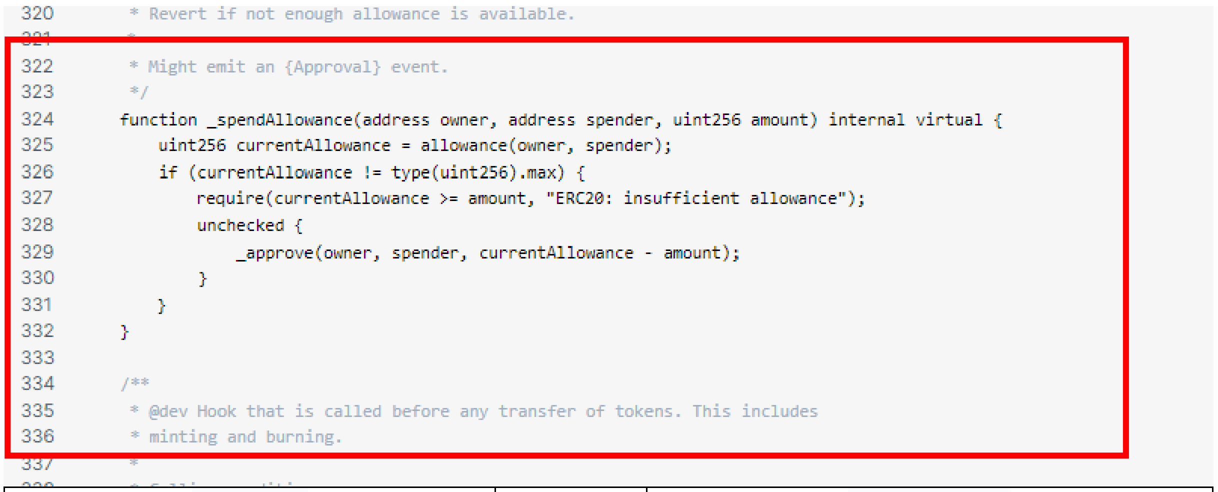
Function	Severity	Remedation
Some tokens do not revert the transaction when the transfer or transferFrom fails and returns False. Hence we must check the return value after calling the transfer or transferFrom function.	Severity : High	Use OpenZeppelin SafeERC20's safetransfer and safetransferFrom functions.

⚠ Approve Front Running Attack (2 Item)



```
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        address owner = _msgSender();
        _approve(owner, spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
```

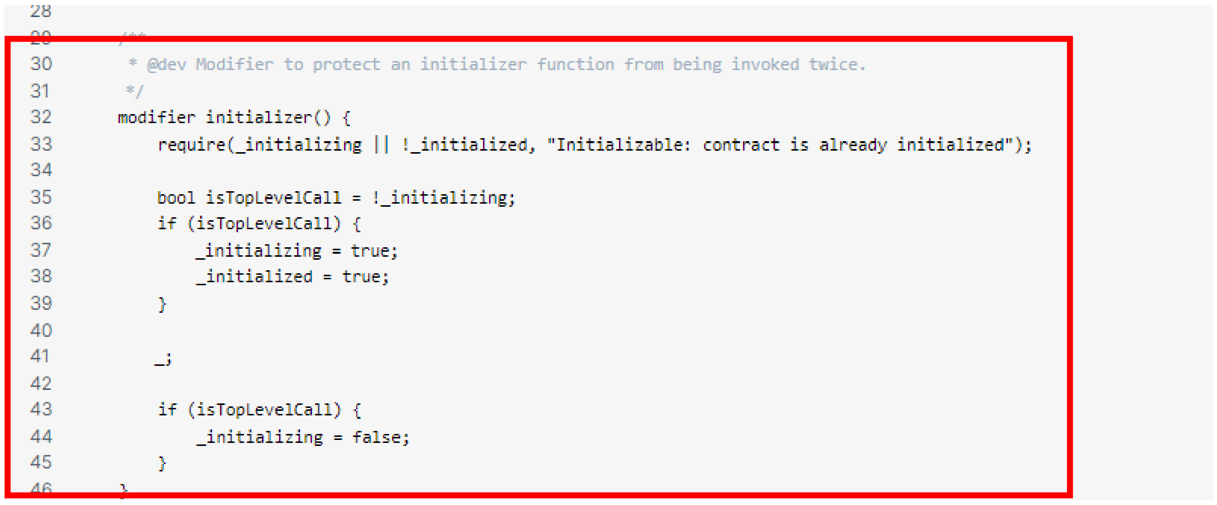
Function	Severity	Remedation
The approve() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by 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.



```
     * Revert if not enough allowance is available.
     *
     * Might emit an {Approval} event.
     */
    function _spendAllowance(address owner, address spender, uint256 amount) internal virtual {
        uint256 currentAllowance = allowance(owner, spender);
        if (currentAllowance != type(uint256).max) {
            require(currentAllowance >= amount, "ERC20: insufficient allowance");
            unchecked {
                _approve(owner, spender, currentAllowance - amount);
            }
        }
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
```

Function	Severity	Remedation
The _spendAllowance() 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 _spendAllowance 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.

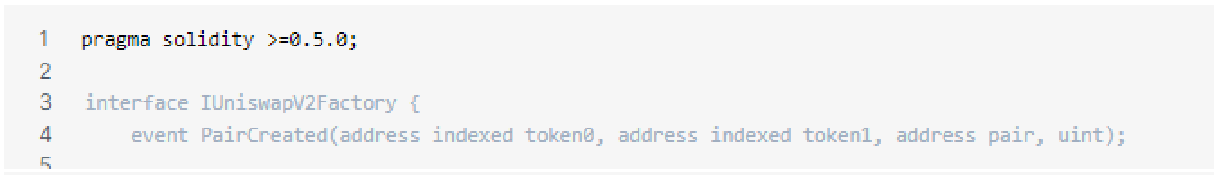
⚠ Modifier Side Effects (1 Item)



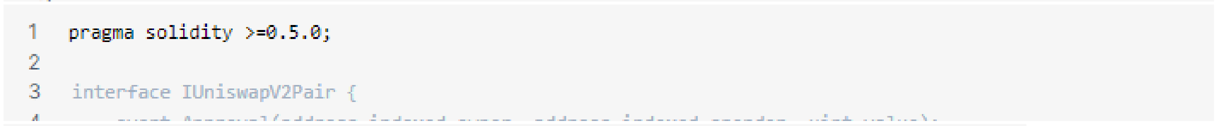
```
    /**
     * @dev Modifier to protect an initializer function from being invoked twice.
     */
    modifier initializer() {
        require(_initializing || !_initialized, "Initializable: contract is already initialized");

        bool isTopLevelCall = !_initializing;
        if (isTopLevelCall) {
            _initializing = true;
            _initialized = true;
        }

        _;

        if (isTopLevelCall) {
            _initializing = false;
        }
    }
```

Function	Severity	Remedation
Solidity functions should always use the Checks-Effects-Interactions pattern which states that the initial stage will contain only checks and validations which resides in the modifiers. Due to this reason, modifiers should only implement checks and validations inside of it and should not make state changes and external calls. The contract Initializable was found to be violating this pattern and the modifier initializer was making sensitive state changes and modifications.	Severity : Medium	Only use modifiers for implementing checks and validations. Do not make external calls or state changing actions inside modifiers.

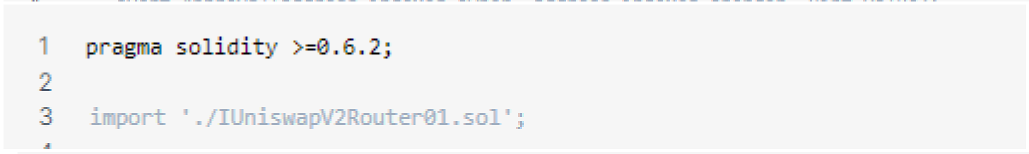
⚠ Use of Multiple Pragma Versions (1 Item)



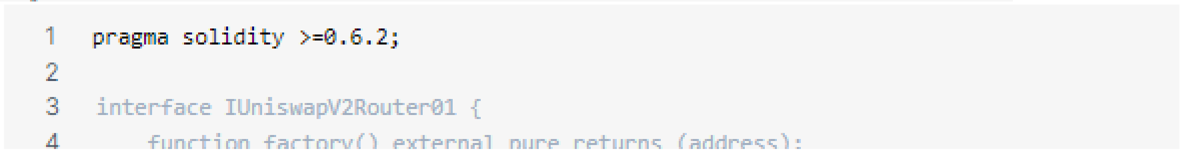
```
pragma solidity >=0.5.0;

interface IUniswapV2Factory {
    event PairCreated(address indexed token0, address indexed token1, address pair, uint);
```



```
pragma solidity >=0.5.0;

interface IUniswapV2Pair {
```



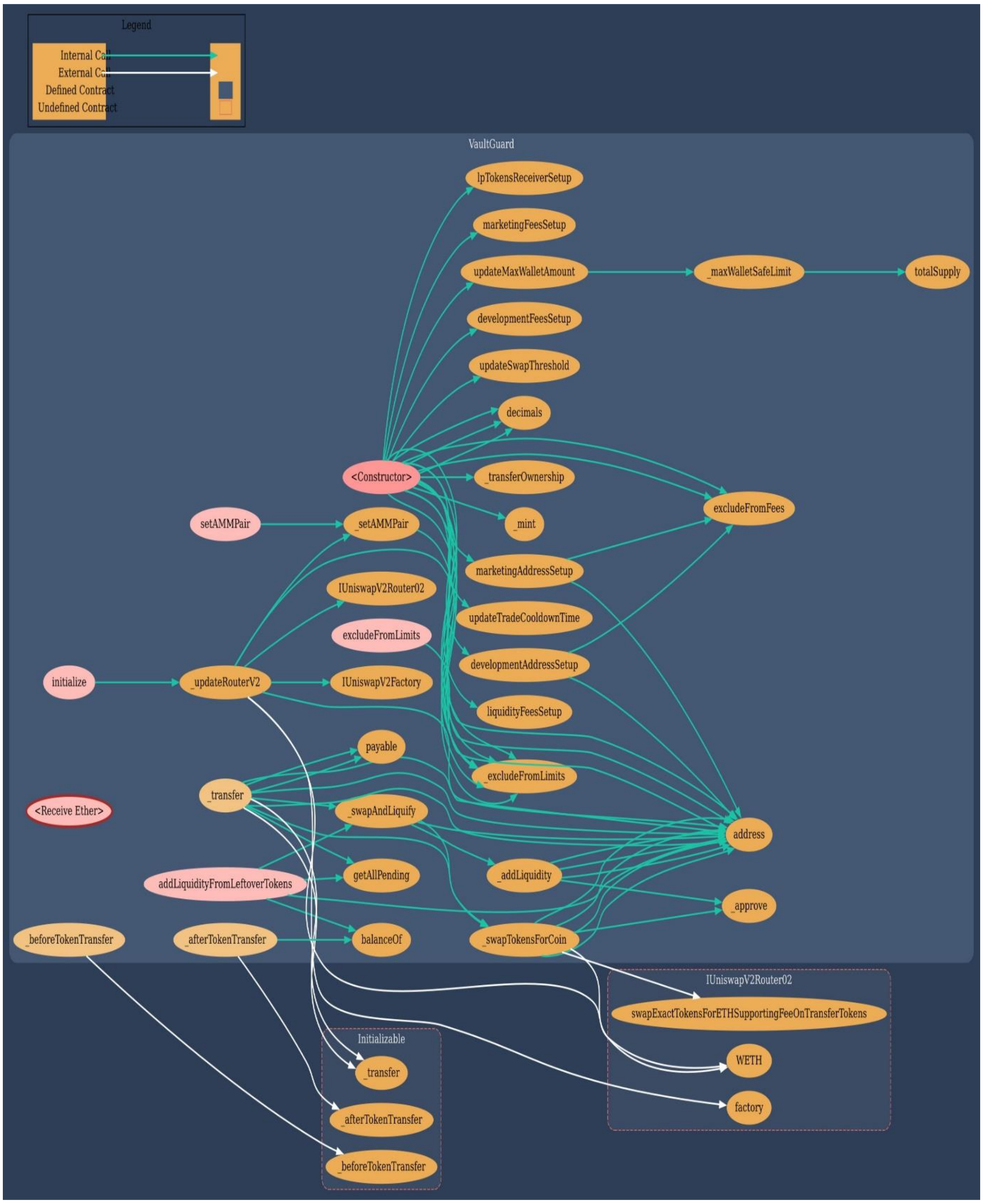
```
pragma solidity >=0.6.2;

import './IUniswapV2Router01.sol';
```



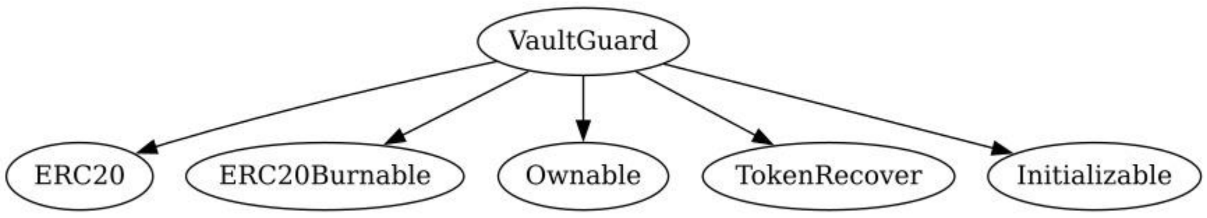
```
pragma solidity >=0.6.2;

interface IUniswapV2Router01 {
    function factory() external pure returns (address);
```

Function	Severity	Remedation
Solidity source files indicate the versions of the compiler they can be compiled with using a pragma directive at the top of the solidity file. This can either be a floating pragma or a specific compiler version. The project was found to be using multiple pragma versions across files which are not considered safe as they can be compiled with all the versions described.	Severity : Low	>=0.6.2 and pragma version and >=0.5.0 pragma version was found in the above solidity files. We recommend updating the Solidity version to 0.8.18 which is considered stable and time-tested. It is also recommended to keep only one version of Solidity across all the contracts.

Contract Flow Graph



Inheritance Graph



Contract Descriptions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
VaultGuard	Implementation	ERC20, ERC20Burnable, Ownable, TokenRecover, Initializable		
		Public ❗	🛑	ERC20
	initialize	External ❗	🛑	initializer
		External ❗	💵	NO❗
	decimals	Public ❗		NO❗
	_swapTokensForCoin	Private 🔐	🛑	
	updateSwapThreshold	Public ❗	🛑	onlyOwner
	getAllPending	Public ❗		NO❗
	developmentAddressSetup	Public ❗	🛑	onlyOwner
	developmentFeesSetup	Public ❗	🛑	onlyOwner
	marketingAddressSetup	Public ❗	🛑	onlyOwner
	marketingFeesSetup	Public ❗	🛑	onlyOwner
	_swapAndLiquify	Private 🔐	🛑	
	_addLiquidity	Private 🔐	🛑	
	addLiquidityFromLeftoverTokens	External ❗	🛑	onlyOwner
	lpTokensReceiverSetup	Public ❗	🛑	onlyOwner

	liquidityFeesSetup	Public ❗	🛑	onlyOwner
	excludeFromFees	Public ❗	🛑	onlyOwner
	_transfer	Internal 🔒	🛑	
	_updateRouterV2	Private 🔐	🛑	
	setAMMPair	External ❗	🛑	onlyOwner
	_setAMMPair	Private 🔐	🛑	
	excludeFromLimits	External ❗	🛑	onlyOwner
	_excludeFromLimits	Internal 🔒	🛑	
	updateMaxWalletAmount	Public ❗	🛑	onlyOwner
	_maxWalletSafeLimit	Private 🔐		
	updateTradeCooldownTime	Public ❗	🛑	onlyOwner
	_beforeTokenTransfer	Internal 🔒	🛑	
	_afterTokenTransfer	Internal 🔒	🛑	

🛑	Function can modify state	💵	Function is payable

Source:

File Name SHA-1 Hash

c:\Solidity\vaultguard.sol de073a5a03a546819dff8c35914a0876af383728

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