CERTIK

Security Assessment

Vemate

Jul 26th, 2022

Table of Contents

Summary

This report has been prepared for Vemate to discover issues and vulnerabilities in the source code of the Vemate project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Vemate
Platform	BSC
Language	Solidity
Codebase	https://bscscan.com/address/0x1f1855A2CeE5FD8Af65446d2ac01FFe458d924b9 https://github.com/kausar75/vemate_token https://bscscan.com/address/0xB33A63e3C5a7055c8E85FfE8eB55Cb9ac65109bD https://github.com/abu-kausar/Vemate-Token/tree/Cerik-Resolved
Commit	c55fd618892a47ec40e875ceb0d1b67a444532d8 8b2e27187397031521f6bc07e3967db5dc00acf9

Audit Summary

Delivery Date	Jul 26, 2022 UTC
Audit Methodology	Static Analysis, Manual Review

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
Critical	0	0	0	0	0	0	0
Major	2	0	0	0	2	0	0
Medium	0	0	0	0	0	0	0
Minor	4	0	0	0	0	0	4
Informational	16	0	0	0	0	0	16
Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
VEM	Vemate.sol	7a4797d14c3c14c9cffc8a150bfc8f7b78440a58935778b29b8b3626dde763e9

Findings

Critical	**0** (0.00%)
Major	**2** (9.09%)
Medium	**0** (0.00%)
Minor	**4** (18.18%)
Informational	**16** (72.73%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
VEM-01	Centralization Risks In Vemate.sol	**Centralization / Privilege**	**Major**	Mitigated
VEM-02	Initial Owner Balance Is The `_totalSupply`	**Centralization / Privilege**	**Major**	Mitigated
VEM-03	Missing Zero Address Validation	Volatile Code	Minor	Resolved
VEM-04	Unused Return Value	Volatile Code	Minor	Resolved
VEM-05	Missing Input Validation	Volatile Code	Minor	Resolved
VEM-06	Function `setSwapTolerancePercent()` Allows For High Slippage	Volatile Code	Minor	Resolved
VEM-13	Comparison To Boolean Constant	Coding Style	Informational	Resolved
VEM-14	Declaration Naming Convention	Coding Style	Informational	Resolved
VEM-15	Function Initializing State	Volatile Code	Informational	Resolved
VEM-16	Shadowing Local Variable	Coding Style	Informational	Resolved
VEM-17	Too Many Digits	Coding Style	Informational	Resolved
VEM-18	Usage Of `block.timestamp`	Language Specific	Informational	Resolved
VEM-19	Missing Emit Events	Coding Style	Informational	Resolved

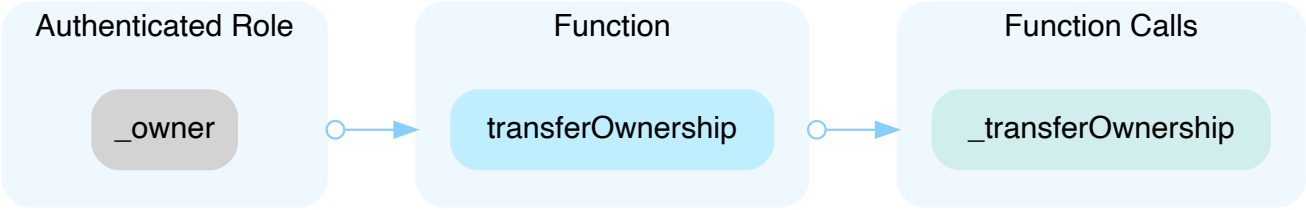
ID	Title	Category	Severity	Status
VEM-20	Inconsistent Comment And Code	Inconsistency	Informational	Resolved
VEM-21	Unnecessary `require` Statement	Coding Style	Informational	Resolved
VEM-22	Use `_msgSender()` From `Ownable`	Coding Style	Informational	Resolved
VEM-23	Inconsistency In `lockedBetweenSells` And `lockedBetweenBuys` Requirements	Inconsistency	Informational	Resolved
VEM-24	`maxTxAmount` Initialized At `_totalSupply` Amount	Volatile Code	Informational	Resolved
VEM-25	Function And Variable Naming Doesn't Match The Operating Environment	Coding Style	Informational	Resolved
VEM-26	Commented Out Code	Coding Style	Informational	Resolved
VEM-27	Antibot Mechanism	Control Flow	Informational	Resolved
VEM-28	Changes To Functionality In Contract Update	Coding Style	Informational	Resolved

VEM-01 | Centralization Risks In Vemate.sol

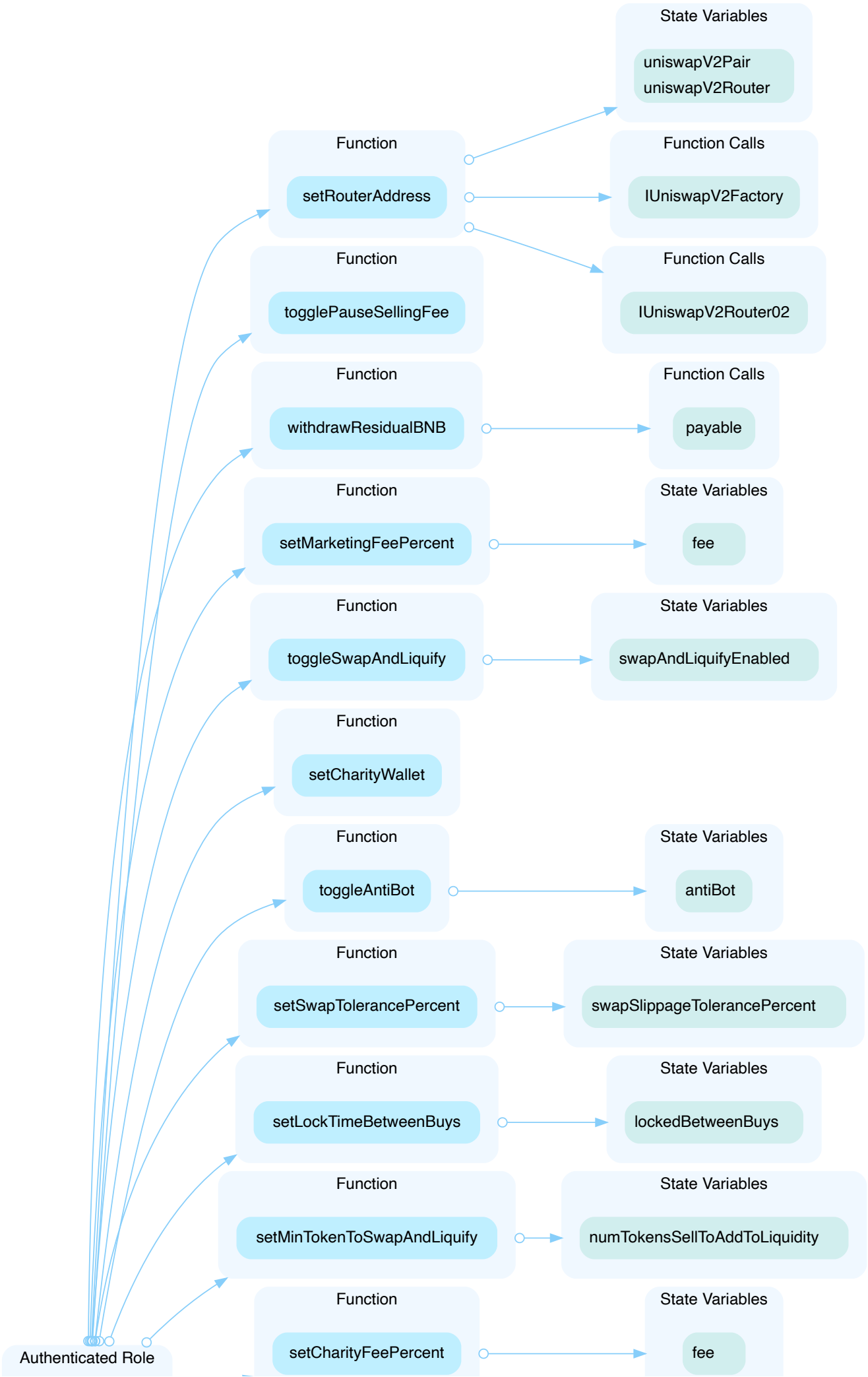
Category	Severity	Location	Status
Centralization / Privilege	Major	Vemate.sol: 139, 701, 717, 728, 739, 750, 758, 769, 780, 791, 802, 813, 818, 823, 830, 837, 842, 848, 853, 858, 865, 871, 875	Mitigated

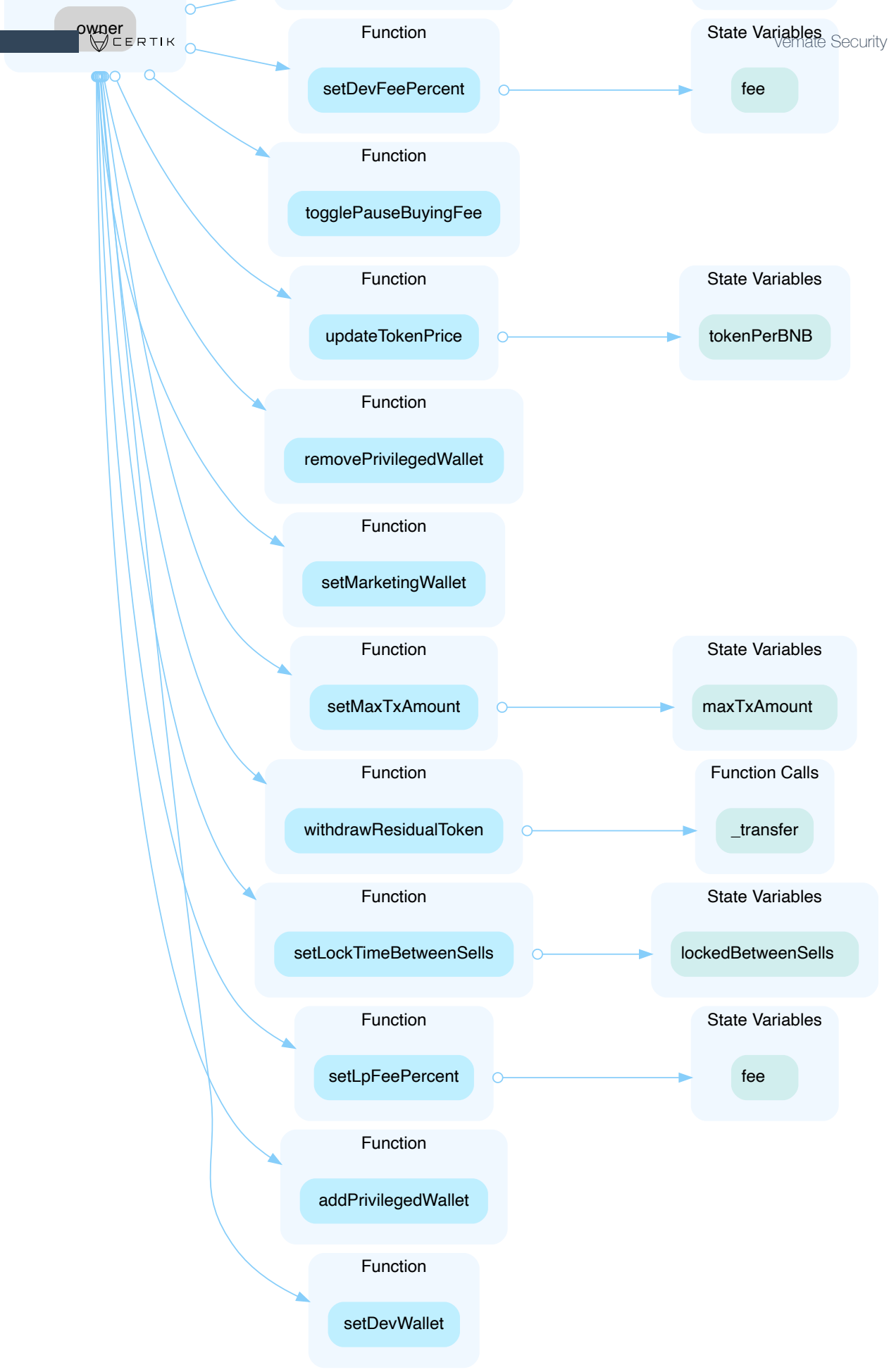
Description

In the contract `Ownable` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and change the address of `_owner` to another contract or user.



In the contract `Vemate` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and completely change features of this contract, such as adjusting fee amounts, draining the funds (in BNB and the contract token) from the contract and sending them to their preferred address, adjusting timelock settings, maliciously updating `maxTxAmount`, `tokenPerBNB`, or `setMinTokenToSwapAndLiquify` to undesirable values. The attacker would also be able to change the router address, adjust the `swapSlippageTolerancePercent` to be as high as 100%, and update the `_isPrivileged` status of any address they want.


State Variables
uniswapV2Pair
uniswapV2Router
Function
setRouterAddress
Function Calls
IUniswapV2Factory
Function
togglePauseSellingFee
Function Calls
IUniswapV2Router02
Function
withdrawResidualBNB
Function Calls
payable
Function
setMarketingFeePercent
State Variables
fee
Function
toggleSwapAndLiquify
State Variables
swapAndLiquifyEnabled
Function
setCharityWallet
Function
toggleAntiBot
State Variables
antiBot
Function
setSwapTolerancePercent
State Variables
swapSlippageTolerancePercent
Function
setLockTimeBetweenBuys
State Variables
lockedBetweenBuys
Function
setMinTokenToSwapAndLiquify
State Variables
numTokensSellToAddToLiquidity
Function
setCharityFeePercent
State Variables
fee
Authenticated Role


owner
Function
setDevFeePercent
State Variables
fee
Function
togglePauseBuyingFee
Function
updateTokenPrice
State Variables
tokenPerBNB
Function
removePrivilegedWallet
Function
setMarketingWallet
Function
setMaxTxAmount
State Variables
maxTxAmount
Function
withdrawResidualToken
Function Calls
_transfer
Function
setLockTimeBetweenSells
State Variables
lockedBetweenSells
Function
setLpFeePercent
State Variables
fee
Function
addPrivilegedWallet
Function
setDevWallet

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*. As such, the team might consider adding back in the `renounceOwnership()` function in the `Ownable` contract.

- Renounce the ownership and never claim back the privileged roles.
  OR

- Remove the risky functionality.

Alleviation

`[CertiK]`: The client has taken several steps towards completing the short-term recommendation for this finding at this time. The changes include:

- 2/3 multi-signature.
  - Multisignature Account: 0x7420bec08C03A9A436B143464009Ea6A43B518DD
  - Authorizing addresses:
    - 0x1537a76331C72A8E43021604B3c633b5A896447a
    - 0x462d99E11749628CafF5B16EcB0bA8815B62594d
    - 0x30D035BdB889AA505e699e4DB8935Cbf55B7BA1C
  - Transaction transferring ownership to multisignature proxy: 0xd3b1e65d81859bc8f0084621e93275798f9eadd831764b28987f600d82164ed5
- Medium post containing this information for the public: https://vemate.medium.com/passing-a-certik-audit-insights-8e07b8ec4617

Additionally, to aid in transparency, the client has undergone KYC: KYC Certification

The team plans to add a time-lock and include the contract in a blog post. Once this is completed, the finding will be updated to mitigated.

`[CertiK]`: In addition to multisig and medium post effort from vemate team, the timelock is also adopted. The timelock is deployed at https://bscscan.com/address/0x1666f214c6d0dbf8be35f5f67fd28bad7da5a482 and the ownership of the vemate deployment at https://bscscan.com/address/0xB33A63e3C5a7055c8E85FfE8eB55Cb9ac65109bD has been transferred into the timelock deployment.

VEM-02 | Initial Owner Balance Is The `_totalSupply`

Category	Severity	Location	Status
Centralization / Privilege	**Major**	Vemate.sol: 696~697	Mitigated

Description

Upon deployment of the contract, the `_totalSupply` for the contract token is held solely by the `_owner` of the contract. This could be a centralization risk as the deployer can distribute tokens without obtaining the consensus of the community.

Recommendation

We recommend the team be transparent regarding the initial token distribution process.

Alleviation

`[Vemate]`: We have resolved this issue by carrying out the following actions:

- Vested full 10% Marketing wallet, with 15% to be released upon listing and then 7% monthly;
- Vested full 9.5% Reserve for CEX listing wallet, with 20% to be released upon listing and then 7% monthly with 10% in the last month;
- Locked full 8% Team tokens for 12 months
- Locked 35% Presale tokens, 20% Token Liquidity tokens as well as 1.1% from Staking, Referral, Partnership and Airdrop wallet tokens until our planned Presale date. This was done via Pinklock.
- Allocated 10% Private Sale tokens plus 1.34% from Staking, Referral, Partnership and Airdrop wallet tokens to Private Sale users - this was allocated as per the selection made by the users - vesting/staking to the Private sale Contract.
- Please find the Private Sale Contract address here, where we have also locked and vested the above mentioned tokens: 0x6C0B36E65026AA6B2DE96f2046Fe3BdDBF6a0e8b

VEM-03 | Missing Zero Address Validation

Category	Severity	Location	Status
Volatile Code	Minor	Vemate.sol: 872	Resolved

Description

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

File: Vemate.sol (Line 872, Function `Vemate.withdrawResidualBNB`)

```
        payable(newAddress).transfer(address(this).balance);
```

- `newAddress` is not zero-checked before being used.

Recommendation

We advise adding a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in commit [c55fd618892a47ec40e875ceb0d1b67a444532d8]

VEM-04 | Unused Return Value

Category	Severity	Location	Status
Volatile Code	Minor	Vemate.sol: 1141~1148	Resolved

Description

The return value of an external call is not stored in a local or state variable.

File: Vemate.sol (Line 1141-1148, Function `Vemate.addLiquidity`)

```
        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            minTokenAmount,
            minETHAmount,
            address(this),
            getCurrentTime()
        );
```

Recommendation

We recommend checking or using the return values of all external function calls. You can additionally add the return values into the event `LiquidityAdded`.

Alleviation

`[CertiK]`: The team removed the `addLiquidityETH()` function in the `VemateToken.sol` in commit c55fd618892a47ec40e875ceb0d1b67a444532d8, which also removes the deflationary mechanism of the token. As such, there is no longer an unchecked return value.

VEM-05 | Missing Input Validation

Category	Severity	Location	Status
Volatile Code	Minor	Vemate.sol: 842, 848, 865	Resolved

Description

For the functions `setMaxTxAmount()`, `updateTokenPrice()`, and `setMinTokenToSwapAndLiquify()`, there is no validation check in place to verify that the input for updated variable meets an expected standard. Since all three functions accept type `uint256`, the input can be any number within this range, which could make the contract behave unexpectedly. For instance, it is possible the `numTokensSellToAddToLiquidity` could be set to 0, meaning a transfer of any amount causes `swapAndLiquify()` to be called. Additionally, it's also possible to input a value for `maxTxAmount` that is greater than the `_totalSupply` value, which would not make sense.

Recommendation

We recommend adding in a validation check that includes the absolute minimum and maximum bounds the team expects to use for the updated inputs.

Alleviation

`[CertiK]`: The team heeded the advice and removed the `updateTokenPrice()` and `setMaxTxAmount()` in the `VemateToken.sol` in the commit c55fd618892a47ec40e875ceb0d1b67a444532d8

VEM-06 | Function `setSwapTolerancePercent()` Allows For High Slippage

Category	Severity	Location	Status
Volatile Code	Minor	Vemate.sol: 858~859	Resolved

Description

The function `setSwapTolerancePercent()` allows for the storage variable `swapSlippageTolerancePercent` to be set to as high as 100%. If it is set to this maximum value, users could lose 100% of their expected BNB output to slippage.

Recommendation

We recommend that the bounds for the input value `newTolerancePercent` be set to a stricter range to avoid the scenario described above.

Alleviation

`[CertiK]`: The team heeded the advice and removed the `setSwapTolerancePercent()` in the `VemateToken.sol` in the commit c55fd618892a47ec40e875ceb0d1b67a444532d8

VEM-13 | Comparison To Boolean Constant

Category	Severity	Location	Status
Coding Style	Informational	Vemate.sol: 752, 759	Resolved

Description

Boolean constants can be used directly and do not need to be compared to true or false.

File: Vemate.sol (Line 752, Function `Vemate.addPrivilegedWallet`)

```
        require(_isPrivileged[newPrivilegedAddress] != true, "already privileged");
```

File: Vemate.sol (Line 759, Function `Vemate.removePrivilegedWallet`)

```
        require(_isPrivileged[prevPrivilegedAddress] != false, "not privileged address");
```

Recommendation

We recommend removing the equality to the boolean constant.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in the commit [c55fd618892a47ec40e875ceb0d1b67a444532d8]

VEM-14 | Declaration Naming Convention

Category	Severity	Location	Status
Coding Style	Informational	Vemate.sol: 640	Resolved

Description

One or more declarations do not conform to the Solidity style guide with regards to its naming convention.

Particularly:

- `camelCase`: Should be applied to function names, argument names, local and state variable names, modifiers
- `UPPER_CASE`: Should be applied to `constant` variables
- `CapWords`: Should be applied to contract names, struct names, event names and enums

---

File: Vemate.sol (Line 640, Contract `Vemate`)

```
    uint8   public constant maxFeePercent = 5;
```

- Constant variable `maxFeePercent` is not in `UPPER_CASE`.

Recommendation

We recommend adjusting those variable and function names to properly conform to Solidity's naming convention.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in the commit c55fd618892a47ec40e875ceb0d1b67a444532d8

VEM-15 | Function Initializing State

Category	Severity	Location	Status
Volatile Code	Informational	Vemate.sol: 649, 658, 659	Resolved

Description

State variables are inline initialized using either non-constant state variable or function calls that are not pure/constant. Since inline initialization occurs before constructor call, some non-constant state variables may not be initialized and non-constant functions may behave in an unexpected way.

File: Vemate.sol (Line 649, Contract `Vemate`)

```
    uint256 private _totalSupply = 150000000 * 10**_decimals; // 150 million;
```

File: Vemate.sol (Line 658, Contract `Vemate`)

```
    uint256 public maxTxAmount = _totalSupply;
```

File: Vemate.sol (Line 659, Contract `Vemate`)

```
    uint256 public numTokensSellToAddToLiquidity = 10000 * 10**_decimals; // 10 Token
```

Recommendation

We recommend removing any inline initialization of state variables via non-constant state variables or non-constant function calls. For these cases, it appears the problem could be resolved by making initializing `_decimals` and `_totalSupply` as constants. Otherwise, if variables must be set upon contract deployment, initialize them in the constructor, instead.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in the commit [c55fd618892a47ec40e875ceb0d1b67a444532d8]

VEM-16 | Shadowing Local Variable

Category	Severity	Location	Status
Coding Style	Informational	Vemate.sol: 937, 1186	Resolved

Description

A local variable is shadowing another component defined elsewhere.

File: Vemate.sol (Line 1186, Function `Vemate._approve`)

```
    function _approve(address owner, address spender, uint256 amount) internal {
```

- Local variable `owner` shadows the function `owner` in `Ownable`.

File: Vemate.sol (Line 123, Contract `Ownable`)

```
    function owner() public view virtual returns (address) {
```

File: Vemate.sol (Line 937, Function `Vemate.allowance`)

```
    function allowance(address owner, address spender) external override view returns
(uint256) {
```

- Local variable `owner` shadows the function `owner` in `Ownable`.

File: Vemate.sol (Line 123, Contract `Ownable`)

```
    function owner() public view virtual returns (address) {
```

Recommendation

We recommend removing or renaming the local variable that shadows another definition. For instance, `owner_` could be used.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in the commit c55fd618892a47ec40e875ceb0d1b67a444532d8

VEM-17 | Too Many Digits

Category	Severity	Location	Status
Coding Style	Informational	Vemate.sol: 649	Resolved

Description

Literals with many digits are difficult to read and review.

File: Vemate.sol (Line 649, Function `Vemate.slitherConstructorVariables`)

```
    uint256 private _totalSupply = 150000000 * 10**_decimals; // 150 million;
```

Recommendation

We advise the client to use the scientific notation to improve readability.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in the commit [c55fd618892a47ec40e875ceb0d1b67a444532d8]

VEM-18 | Usage Of `block.timestamp`

Category	Severity	Location	Status
Language Specific	Informational	Vemate.sol: 1198~1199	Resolved

Description

`block.timestamp` is used for comparison, which can be risky since timestamp can be influenced by miners. That means the miner creating the block can manipulate the `block.timestamp`, to some degree, and change the outcome of the transaction.

File: Vemate.sol (Line 1198-1199, Function `Vemate.checkSwapFrequency`)

```
            require(currentTime - lastSwapTime >= lockedBetweenSells, "Lock time has not
been released from last swap"
            );
```

Recommendation

We recommend against relying on `block.timestamp`.

Reference: https://swcregistry.io/docs/SWC-116

Alleviation

`[CertiK]`: The team removed the function `checkSwapFrequency()` in the `VemateToken.sol` in the commit c55fd618892a47ec40e875ceb0d1b67a444532d8.

VEM-19 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	Informational	Vemate.sol: 871, 875, 1070	Resolved

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles or when important updates are made to the contract.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles, or when important updates are made to the contract, like receiving BNB.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in commit [c55fd618892a47ec40e875ceb0d1b67a444532d8]

VEM-20 | Inconsistent Comment And Code

Category	Severity	Location	Status
Inconsistency	Informational	Vemate.sol: 659	Resolved

Description

The comment in line 659 looks like it is saying that `numTokensSellToAddToLiquidity` should be 10, but instead, this value is set to 10,000.

Recommendation

Please ensure the value is correctly set and update the comment as needed.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in the commit [c55fd618892a47ec40e875ceb0d1b67a444532d8]

VEM-21 | Unnecessary `require` Statement

Category	Severity	Location	Status
Coding Style	Informational	Vemate.sol: 677	Resolved

Description

```
require(owner() != address(0), "Owner must be set");
```

The `require` code here is unnecessary since, upon construction, the owner is set to the address that deploys the contract. It is not possible for `address(0)` to deploy a contract.

Recommendation

This line may be safely omitted.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in commit [c55fd618892a47ec40e875ceb0d1b67a444532d8]

VEM-22 | Use `_msgSender()` From `Ownable`

Category	Severity	Location	Status
Coding Style	Informational	Vemate.sol: 698	Resolved

Description

In the event `Transfer`, `msg.sender` is used rather than the inherited function `_msgSender()`.

Recommendation

We recommend using `_msgSender()` in place of `msg.sender` for consistency throughout the contract. Consistency promotes readability in the contract.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in commit [c55fd618892a47ec40e875ceb0d1b67a444532d8]

VEM-23 | Inconsistency In `lockedBetweenSells` And `lockedBetweenBuys` Requirements

Category	Severity	Location	Status
Inconsistency	Informational	Vemate.sol: 656, 657, 824~825, 831	Resolved

Description

The storage variable `lockedBetweenSells` is initialized at 60, but can be updated with the function, `setLockTimeBetweenSells()`. This function requires that the new input for `lockedBetweenSells` is no more than 30.

Recommendation

While this is conceivable as a strategy at launch, please confirm this is the intention of the team. We recommend reviewing the initialized value and bound on the new input and make any necessary changes if this is an inconsistency error.

Alleviation

`[CertiK]`: The team has heeded the advice by reviewing the locking mechanism and has opted to remove the mechanism and related variables from the contract.

VEM-24 | `maxTxAmount` Initialized At `_totalSupply` Amount

Category	Severity	Location	Status
Volatile Code	Informational	Vemate.sol: 658	Resolved

Description

The storage variable `maxTxAmount` is initialized inline to be equivalent to the `_totalSupply` of contract tokens. Effectively, this means that at initialization, there is no bound on the number of tokens that can be transferred in a single transaction.

Recommendation

If this is the intended effect, where any user can send any amount at launch start, there is no need to make a change. However, if this is initalized at `_totalSupply` because the `_owner` of the contract intends to move the supply from their contract balance, it does not seem necessary to initialize `maxTxAmount` equal to `_totalSupply`, since the `_owner` is a privileged address and its transfers are not checked against `maxTxAmount` anyway. The client team might consider initializing `maxTxAmount` at the value they intend to update it with.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in the commit [c55fd618892a47ec40e875ceb0d1b67a444532d8]

VEM-25 | Function And Variable Naming Doesn't Match The Operating Environment

Category	Severity	Location	Status
Coding Style	Informational	Vemate.sol: 1106~1107, 1114, 1116, 1121, 1132, 1137, 1145, 1149	Resolved

Description

The `Vemate` contract switches between referencing ETH and BNB within multiple functions, variables, and comments. The following function declared within the `Vemate` contract includes references to ETH:

- `swapTokensForEth()`

The following local variables include references to ETH:

- `ethAmount`
- `minETHAmount`

Recommendation

We recommend uniformly making references to BNB instead of ETH for any functions, variables, or comments declared within this contract, including but not limited to those listed in the description.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in the commit [c55fd618892a47ec40e875ceb0d1b67a444532d8]

VEM-26 | Commented Out Code

Category	Severity	Location	Status
Coding Style	Informational	Vemate.sol: 1133~1134	Resolved

Description

The commented out code is not relevant to the function, `addLiquidity()`.

Recommendation

We recommend removing the commented out line of code.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in the commit c55fd618892a47ec40e875ceb0d1b67a444532d8

VEM-27 | Antibot Mechanism

Category	Severity	Location	Status
Control Flow	Informational	Vemate.sol: 1194~1195	Resolved

Description

The function `checkSwapFrequency()` only checks the address of the `msg.sender`, which can be a contract or an externally owned account. It is easy for a user to set up interactions between multiple contracts and bypass this anti-bot feature by initiating from one contract, using that contract to call a separate contract which then interacts with the `Vemate` contract successfully within the time-lock since it does not have the same address. With this in mind, the `checkSwapFrequency()` function may not have its intended effect.

Recommendation

We encourage the team to consider this possibility and decide whether to make changes to the function.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in the commit c55fd618892a47ec40e875ceb0d1b67a444532d8

VEM-28 | Changes To Functionality In Contract Update

Category	Severity	Location	Status
Coding Style	Informational	Vemate.sol: 612~613	Resolved

Description

In commit c55fd618892a47ec40e875ceb0d1b67a444532d8, significant changes have been made to the functionality of file `Vemate.sol` outside the recommendations found in the audit. Below is a brief list of notable changes made from the original file.

- In the struct `FeePercent`, the following `uint8` values have been removed: `lp`, `dev`, and `marketing`. The `uint8` value `treasury` has been added.
- The struct outlined above initially had the following fee percent distribution: `lp` was 2, `dev` was 1, `marketing` was 1, and `charity` was 1. In the update, `treasury` is now 4 while `charity` is 1.
- The symbol of the token has been changed from `V` to `VMT`.
- A privileged-status function `setAutomatedMarketMakerPair()` was added. In this function, the owner of the contract can set a new pair address for use in determining fees on transfers
- The internal function `_transfer()` has been altered. The function call to `swapAndLiquify()` only occurs on token sells.
- The function `swapAndLiquify()` has been altered. Most notably, the calculation of fees differs because there is no longer a portion sent to `addLiquidity()`. Instead, the function now takes the entire contract token balance as the amount to swap, rather than the previous threshold amount `numTokensSellToAddToLiquidity`. This amount of tokens is swapped for `BNB` and the entire amount of `BNB` in the contract is divided into fees, 80% going to `treasury` address and 20% going to `charity` address.
- The `addLiquidity()` function has been removed from the contract. This contract no longer has an automated liquidity acquisition feature.

Recommendation

Consider revisit the contract to make sure the changes aligning with the original design.

Alleviation

[`CertiK`]: The Vemate team confirms that all changes listed above are intentional and align with the design of the token.

Optimizations

ID	Title	Category	Severity	Status
VEM-07	Unused State Variable	Gas Optimization	Optimization	Resolved
VEM-08	Variables That Could Be Declared As `constant`	Gas Optimization	Optimization	Resolved
VEM-09	Contract Size Exceeds 24576 Bytes	Compiler Error, Gas Optimization	Optimization	Resolved
VEM-10	Only Update Necessary Storage Variables	Gas Optimization	Optimization	Resolved
VEM-11	`_approve()` Function Call Can Be Unchecked	Gas Optimization	Optimization	Resolved
VEM-12	Update To Sender Balance Can Be Made `unchecked`	Gas Optimization	Optimization	Resolved

VEM-07 | Unused State Variable

Category	Severity	Location	Status
Gas Optimization	Optimization	Vemate.sol: 645	Resolved

Description

The following state variable is never used in the codebase.

Variable `blockTimestampLast` in `Vemate` is never used in `Vemate`.

File: Vemate.sol (Line 645, Contract `Vemate`)

```
    uint32  private blockTimestampLast;
```

Recommendation

We advise removing the unused variables.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in the commit c55fd618892a47ec40e875ceb0d1b67a444532d8

VEM-08 | Variables That Could Be Declared As `constant`

Category	Severity	Location	Status
Gas Optimization	Optimization	Vemate.sol: 635, 636, 639, 649	Resolved

Description

The linked variables could be declared as `constant` since these state variables are never modified.

Recommendation

We recommend to declare these variables as `constant`.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in commit [c55fd618892a47ec40e875ceb0d1b67a444532d8]

VEM-09 | Contract Size Exceeds 24576 Bytes

Category	Severity	Location	Status
Compiler Error, Gas Optimization	Optimization	Vemate.sol: 612	Resolved

Description

`Vemate` contract code size exceeds 24576 bytes, so the contract may not be deployable on the mainnet.

Recommendation

Consider enabling the optimizer (with a low "runs" value), turning off revert strings, or using libraries.

VEM-10 | Only Update Necessary Storage Variables

Category	Severity	Location	Status
Gas Optimization	Optimization	Vemate.sol: 774~775, 785~786, 796~797, 807~808	Resolved

Description

For each of the functions, `setLpFeePercent`, `setDevFeePercent`, `setMarketingFeePercent`, and `setCharityFeePercent`, the whole storage variable `fee` is updated during each execution of the functions, when only one of the objects (`fee.lp`, `fee.dev`, `fee.marketing`, or `fee.charity`) is actually being changed during the function call. Storage variables are expensive to update, and should be minimized whenever possible.

Recommendation

To minimize gas costs, we recommend only updating the necessary storage variables.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in the commit c55fd618892a47ec40e875ceb0d1b67a444532d8

VEM-11 | `_approve()` Function Call Can Be Unchecked

Category	Severity	Location	Status
Gas Optimization	Optimization	Vemate.sol: 970~971, 1009	Resolved

Description

In line 969, the `_currentAllowance` is confirmed to be at least as large as the input `amount` through the `require` check. Hence, the SafeMath library inherent to the solidity compiler version, which protects against underflows and overflows, is not necessary here.

Similarly, in line 1008, `_currentAllowance` is confirmed to be at least as large as the `subtractedValue`.

Recommendation

The `_approve()` function call after each `require` line can be safely declared inside the body of `unchecked{}`.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in commit [c55fd618892a47ec40e875ceb0d1b67a444532d8]

VEM-12 | Update To Sender Balance Can Be Made `unchecked`

Category	Severity	Location	Status
Gas Optimization	Optimization	Vemate.sol: 1168~1169	Resolved

Description

Since there was a check that the input `amount` for this function does not exceed `_balances[sender]` in the internal function `_transfer()`, the update to `_balances[sender]` can be safely declared inside `unchecked{}`. The difference, `_balances[sender] - amount` will not underflow, and this will optimize gas savings.

Recommendation

We recommend writing the aforementioned line in an `unchecked{}` block to temporarily disable `SafeMath`.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the `VemateToken.sol` in the commit [c55fd618892a47ec40e875ceb0d1b67a444532d8]

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.