

Vonder Finance

Token and Farm

Smart Contract Audit Report



Date Issued: 24 Aug 2021

Version: Final v.1.0

Valix
Consulting

Public

Table of Contents

Executive Summary	3
Overview	3
Scope of Work	3
Auditors	4
Disclaimer	4
Audit Result Summary	5
Methodology	6
Audit Items	7
Risk Rating	9
Findings	10
Review Findings Summary	10
Detailed Result	11
Appendix	47
About Us	47
Contact Information	47
References	48

Executive Summary

Overview

Valix conducted a smart contract audit to evaluate potential security issues of the **Token and Farm features**. This audit report was published on *August 24, 2021*. The audit scope is limited to the **Token and Farm features**. Our security best practices strongly recommend that the **Vonder Finance team** conduct a full security audit for both on-chain and off-chain components of its infrastructure and their interaction. A comprehensive examination has been performed during the audit process utilizing Valix's Formal Verification, Static Analysis, and Manual Review techniques.

Scope of Work

The security audit conducted does not replace the full security audit of the overall Vonder Finance protocol. The scope is limited to the **Token and Farm features** and their related smart contracts.

The security audit covered the components at this specific state:

Item	Description
Components	<ul style="list-style-type: none"> ▪ <i>Vonder MasterChef smart contract</i> ▪ <i>VonderToken smart contract</i> ▪ <i>Imported associated smart contracts</i>
GitHub Repository	<ul style="list-style-type: none"> ▪ https://github.com/vonderfinance/vonder-masterchef
Commit	<ul style="list-style-type: none"> ▪ <code>edacb5cb3ed72546e706043bfe3078a63cb07fbe</code>
Reassessment Commit	<ul style="list-style-type: none"> ▪ <code>36dec4e96394925af233ea08c0490e4f18edf3ac</code>
Audited Files	<ul style="list-style-type: none"> ▪ <i>MasterChef.sol</i> ▪ <i>VonderToken.sol</i>
Excluded Files/Contracts	-

Remark: Our security best practices strongly recommend that the Vonder Finance team conduct a full security audit for both on-chain and off-chain components of its infrastructure and the interaction between them.

Auditors

Phuwanai Thummavet
Sumedt Jitpukdebonin
Keerati Torach
Boonpoj Thongakaraniroj

Disclaimer

Our smart contract audit was conducted over a limited period and was performed on the smart contract at a single point in time. As such, the scope was limited to current known risks during the work period. The review does not indicate that the smart contract and blockchain software has no vulnerability exposure.

We reviewed the security of the smart contracts with our best effort, and we do not guarantee a hundred percent coverage of the underlying risk existing in the ecosystem. The audit was scoped only in the provided code repository. The on-chain code is not in the scope of auditing.

This audit report does not provide any warranty or guarantee, nor should it be considered an “approval” or “endorsement” of any particular project. This audit report should also not be used as investment advice nor provide any legal compliance.

Audit Result Summary

From the audit results and the remediation and response from the developer, Valix trusts that the **Token and Farm features** have sufficient security protections to be safe for use.



Initially, Valix was able to identify **14 issues** that were categorized from the “Critical” to “Informational” risk level in the given timeframe of the assessment. On the reassessment, all high and medium risk issues were fixed. For the acknowledged issues, the Vonder team acknowledged each issue but decided to remain the original code. Below is the breakdown of the vulnerabilities found and their associated risk rating for each assessment conducted.

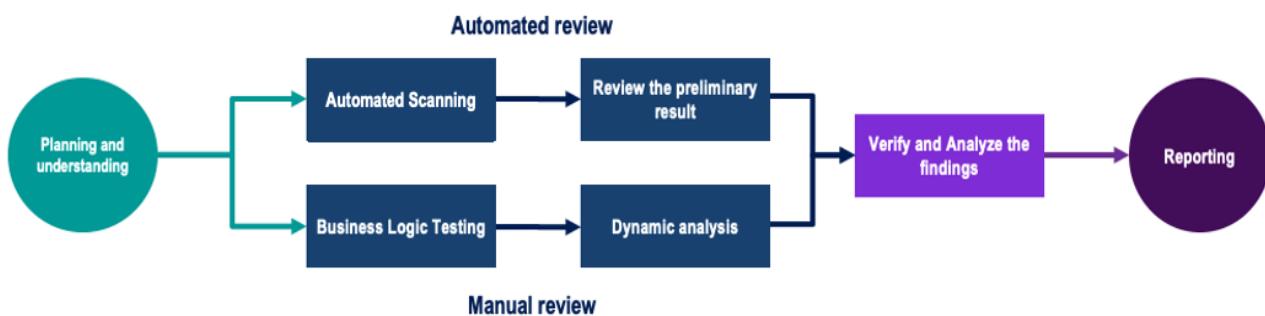
Target	Assessment Result					Reassessment Result				
	C	H	M	L	I	C	H	M	L	I
Token and Farm	-	4	2	5	3	-	0	0	5	3

Note: Risk Rating

C Critical, **H** High, **M** Medium, **L** Low, **I** Informational

Methodology

The smart contract security audit methodology is based on Smart Contract Weakness Classification and Test Cases (SWC Registry), CWE, well-known best practices, and smart contract hacking case studies. Manual and automated review approaches can be mixed and matched, including business logic analysis in terms of the malicious doer's perspective. Using automated scanning tools to navigate or find offending software patterns in the codebase along with a purely manual or semi-automated approach, where the analyst primarily relies on one's knowledge, is performed to eliminate the false-positive results.



Planning and Understanding

- Determine the scope of testing and understanding the application's purposes and workflows.
- Identify key risk areas, including technical and business risks.
- Determine which sections to review within the resource constraints and review method – automated, manual or mixed.

Automated Review

- Adjust automated source code review tools to inspect the code for known unsafe coding patterns.
- Verify the tool's output to eliminate false-positive results, and adjust and re-run the code review tool if necessary.

Manual Review

- Analyzing the business logic flaws requires thinking in unconventional methods.
- Identify unsafe coding behavior via static code analysis.

Reporting

- Analyze the root cause of the flaws.
- Recommend improvements for secure source code.

Audit Items

We perform the audit according to the following categories and test names.

Category	ID	Test Name
Security Issue	SEC01	<i>Authorization Through tx.origin</i>
	SEC02	<i>Business Logic Flaw</i>
	SEC03	<i>Delegatecall to Untrusted Callee</i>
	SEC04	<i>DoS With Block Gas Limit</i>
	SEC05	<i>DoS with Failed Call</i>
	SEC06	<i>Function Default Visibility</i>
	SEC07	<i>Hash Collisions With Multiple Variable Length Arguments</i>
	SEC08	<i>Incorrect Constructor Name</i>
	SEC09	<i>Improper Access Control or Authorization</i>
	SEC10	<i>Improper Emergency Response Mechanism</i>
	SEC11	<i>Insufficient Validation of Address Length</i>
	SEC12	<i>Integer Overflow and Underflow</i>
	SEC13	<i>Outdated Compiler Version</i>
	SEC14	<i>Outdated Library Version</i>
	SEC15	<i>Private Data On-Chain</i>
	SEC16	<i>Reentrancy</i>
	SEC17	<i>Transaction Order Dependence</i>
	SEC18	<i>Unchecked Call Return Value</i>
	SEC19	<i>Unexpected Token Balance</i>
	SEC20	<i>Unprotected Assignment of Ownership</i>
	SEC21	<i>Unprotected SELFDESTRUCT Instruction</i>
	SEC22	<i>Unprotected Token Withdrawal</i>
	SEC23	<i>Unsafe Type Inference</i>
	SEC24	<i>Use of Deprecated Solidity Functions</i>
	SEC25	<i>Use of Untrusted Code or Libraries</i>
	SEC26	<i>Weak Sources of Randomness from Chain Attributes</i>
	SEC27	<i>Write to Arbitrary Storage Location</i>

Category	ID	Test Name
Functional Issue	FNC01	<i>Arithmetic Precision</i>
	FNC02	<i>Permanently Locked Fund</i>
	FNC03	<i>Redundant Fallback Function</i>
	FNC04	<i>Timestamp Dependence</i>
Operational Issue	OPT01	<i>Code With No Effects</i>
	OPT02	<i>Message Call with Hardcoded Gas Amount</i>
	OPT03	<i>The Implementation Contract Flow or Value and the Document is Mismatched</i>
	OPT04	<i>The Usage of Excessive Byte Array</i>
	OPT05	<i>Unenforced Timelock on An Upgradeable Proxy Contract</i>
Developmental Issue	DEV01	<i>Assert Violation</i>
	DEV02	<i>Other Compilation Warnings</i>
	DEV03	<i>Presence of Unused Variables</i>
	DEV04	<i>Shadowing State Variables</i>
	DEV05	<i>State Variable Default Visibility</i>
	DEV06	<i>Typographical Error</i>
	DEV07	<i>Uninitialized Storage Pointer</i>
	DEV08	<i>Violation of Solidity Coding Convention</i>
	DEV09	<i>Violation of Token (ERC20) Standard API</i>

Risk Rating

To prioritize the vulnerabilities, we have adopted the scheme of five distinct levels of risk: **Critical**, **High**, **Medium**, **Low**, and **Informational**, based on OWASP Risk Rating Methodology. The risk level definitions are presented in the table.

Risk Level	Definition
Critical	The code implementation does not match the specification, and it could disrupt the platform.
High	The code implementation does not match the specification, or it could result in the loss of funds for contract owners or users.
Medium	The code implementation does not match the specification under certain conditions, or it could affect the security standard by losing access control.
Low	The code implementation does not follow best practices or use suboptimal design patterns, which may lead to security vulnerabilities further down the line.
Informational	Findings in this category are informational and may be further improved by following best practices and guidelines.

The **risk value** of each issue was calculated from the product of the **impact** and **likelihood values**, as illustrated in a two-dimensional matrix below.

- **Likelihood** represents how likely a particular vulnerability is exposed and exploited in the wild.
- **Impact** measures the technical loss and business damage of a successful attack.
- **Risk** demonstrates the overall criticality of the risk.

Impact \ Likelihood	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Informational

The shading of the matrix visualizes the different risk levels. Based on the acceptance criteria, the risk levels "Critical" and "High" are unacceptable. Any issue obtaining the above levels must be resolved to lower the risk to an acceptable level.

Findings

Review Findings Summary

The table below shows the summary of our assessments.

No.	Issue	Risk	Status	Functionality is in use
1	Voting Amplification	High	Fixed	Not in use
2	Voting Displacement	High	Fixed	Not in use
3	Contract Parameters Can Be Altered By The Platform Developer Without Timelock	High	Partially Fixed	In use
4	Redelegation Failure	High	Fixed	Not in use
5	No Maximum Supply Minting Check	Medium	Fixed	In use
6	No LP Token Adding After Deposit	Medium	Fixed	In use
7	Emission Rate Update May Fail	Low	Acknowledged	In use
8	<i>DevAddress</i> Reassignment May Fail	Low	Acknowledged	In use
9	<i>FeeAddress</i> Reassignment May Fail	Low	Acknowledged	In use
10	The Compiler May Be Susceptible To The Publicly Disclosed Bugs	Low	Acknowledged	In use
11	The Compiler Is Not Locked To A Specific Version	Low	Acknowledged	In use
12	Same LP Token May Be Added More Than Once	Informational	Acknowledged	In use
13	The Function Name With Internal Visibility Is Not Complied With The Naming Convention	Informational	Acknowledged	In use
14	Public Functions That Could Be Declared As <i>external</i>	Informational	Acknowledged	In use

The statuses of the issues are defined as follows:

Fixed: The issue has been completely resolved and has no further complications.

Partially Fixed: The issue has been partially resolved.

Acknowledged: The issue's risk has been reported and acknowledged.

Detailed Result

This section provides our issues found in detail.

No. 1	Voting Amplification					
Risk	High	Likelihood	High			
		Impact	Medium			
Functionality is in use	Not in use	Status	Fixed			
Associated Files	<i>VonderToken.sol</i>					
Locations	<i>_moveDelegates(address, address, uint256)</i> L:874 - 892					
Description						
<p>VON token was designed to be a governance token. Therefore, VON token holders can vote on the desired representative or proposal by delegating their tokens to. The <i>_moveDelegates</i> function would be executed during the delegation/voting process to transfer votes (represented by VON tokens) from each delegator to a representative.</p> <p>However, the <i>_moveDelegates</i> function does not lock up the delegated VON tokens inside the contract. This delegation mechanism potentially causes a double-spending issue leading to a Sybil attack which amplifies the voting power improperly.</p> <p>Consider the following voting amplification attack scenario:</p> <ol style="list-style-type: none"> 1. Attacker #1 has 100 tokens and delegates his vote to Bob (the representative). Bob gains 100 votes now. 2. Attacker #1 transfers his 100 tokens to Attacker #2. 3. Attacker #2 delegates the obtained 100 tokens to Bob. Now, Bob captures 200 votes. 4. Attackers can easily amplify Bob's votes by performing Steps 2 and 3 repeatedly. 						
Recommendations						
<p>We recommend two possible solutions. The first solution is improving the <i>VonderToken</i> contract to lock away the delegated VON tokens inside until the voting or delegating period is complete. The <i>VonderToken</i> contract also has to record the number of votes of each delegator correctly so that the contract can check and move each delegator's votes precisely when re-delegating.</p> <p>Another solution is implementing another <i>voting</i> contract and using VON tokens as the contract's voting tokens. The <i>voting</i> contract also needs to lock up and record the delegated VON tokens correctly nonetheless.</p>						

Platform Developer Response

The affected voting functionality was removed from the VonderToken contract.

Detailed Issue

This issue enables attackers to massively amplify their votes on any desired representative or proposals with a minimal attack cost.

VonderToken.sol

```

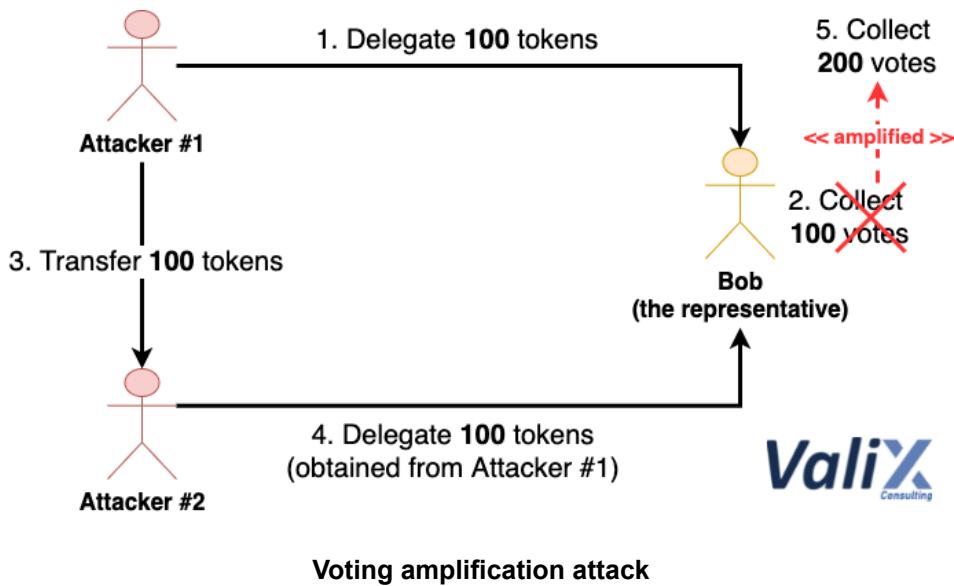
874   function _moveDelegates(address srcRep, address dstRep, uint256 amount) internal
875   {
876     if (srcRep != dstRep && amount > 0) {
877       if (srcRep != address(0)) {
878         // decrease old representative
879         uint32 srcRepNum = numCheckpoints[srcRep];
880         uint256 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum - 1].votes : 0;
881         uint256 srcRepNew = srcRepOld.sub(amount);
882         _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
883       }
884       if (dstRep != address(0)) {
885         // increase new representative
886         uint32 dstRepNum = numCheckpoints[dstRep];
887         uint256 dstRepOld = dstRepNum > 0 ? checkpoints[dstRep][dstRepNum - 1].votes : 0;
888         uint256 dstRepNew = dstRepOld.add(amount);
889         _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
890       }
891     }
892   }

```

The code snippet above shows the `_moveDelegates` function that is the root cause of the issue. This function is executed during the voting delegation process to move the delegator's votes to the representative. In other words, the amount of the votes (represented by the VON tokens) from a delegator will be increased to the representative (line no's. 886–889).

Although the `_moveDelegates` function can move the delegator's votes to the targeting representative correctly, the function does not lock up the delegated VON tokens inside the contract.

This design flaw opens the room for a double-spending attack in which attackers can create Sybil accounts leading to the voting amplification.



Consider the voting amplification attack scenario in the figure above.

1. **Attacker #1** initially has *100 tokens* and delegates his vote to **Bob**
2. **Bob** now collects *100 votes*
3. **Attacker #1** transfers his *100 tokens* to **Attacker #2**
4. **Attacker #2** delegates the obtained *100 tokens* to **Bob**
5. **Bob's** collected votes have been amplified to *200*

The attackers can easily amplify Bob's votes by performing Steps 3 and 4 repeatedly.

Reassessment

The affected voting functionality was removed from the VonderToken contract.

No. 2	Voting Displacement					
Risk	High	Likelihood	High			
Functionality is in use	Not in use	Impact	Medium			
Affected Files	<i>VonderToken.sol</i>					
Locations	<code>_delegate(address, address) L:862 - 872</code> <code>_moveDelegates(address, address, uint256) L:874 - 892</code>					
Description						
<p>A voter/delegator (VON token holder) can re-delegate his votes to another representative by calling the external <code>delegate</code> or <code>delegateBySig</code> function. The external function will subsequently call the internal <code>_delegate</code> function which will obtain a delegator balance.</p> <p>The <code>_moveDelegates</code> function is then invoked to move the delegator's votes (the previously obtained delegator balance) from the current to the new representative. Unfortunately, this redelegation mechanism allows attackers to perform the votes withdrawal over their previous delegation, potentially leading to an attack that displaces other voters' votes.</p>						
Consider the following voting displacement scenario:						
<ol style="list-style-type: none"> 1. Bob (the representative) received 450 votes from other voters. 2. Attacker #1 has 1 token and delegates his vote to Bob. Bob now has 451 votes. 3. Attacker #2 transfers 450 tokens to Attacker #1. Attacker #1 now has 451 tokens in his wallet. 4. Attacker #1 re-delegates his vote from Bob to Attacker #2. Since the current token balance of Attacker #1 is 451, the <code>_moveDelegate</code> function moves 451 votes from Bob to Attacker #2. Total votes from other voters were displaced unexpectedly. 						
Recommendations						
<p>We recommend two possible solutions. The first solution is improving the <i>VonderToken</i> contract to lock away the delegated VON tokens inside until the voting or delegating period is complete. The <i>VonderToken</i> contract also has to record the number of votes of each delegator correctly so that the contract can check and move each delegator's votes precisely when re-delegating.</p> <p>Another solution is implementing another <i>voting</i> contract and using VON tokens as the contract's voting tokens. The <i>voting</i> contract also needs to lock up and record the delegated VON tokens correctly nonetheless.</p>						
Platform Developer Response						
The affected voting functionality was removed from the <i>VonderToken</i> contract.						

Detailed Issue

This issue allows attackers to take out other voters' votes.

VonderToken.sol

```

862 function _delegate(address delegator, address delegatee)
863     internal
864 {
865     address currentDelegate = _delegates[delegator];
866     uint256 delegatorBalance = balanceOf(delegator); // balance of underlying
867     VONs (not scaled);
868     _delegates[delegator] = delegatee;
869
870     emit DelegateChanged(delegator, currentDelegate, delegatee);
871
872     _moveDelegates(currentDelegate, delegatee, delegatorBalance);
873 }
874
875 function _moveDelegates(address srcRep, address dstRep, uint256 amount) internal
876 {
877     if (srcRep != dstRep && amount > 0) {
878         if (srcRep != address(0)) {
879             // decrease old representative
880             uint32 srcRepNum = numCheckpoints[srcRep];
881             uint256 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum -
882             1].votes : 0;
883             uint256 srcRepNew = srcRepOld.sub(amount);
884             _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
885         }
886
887         if (dstRep != address(0)) {
888             // increase new representative
889             uint32 dstRepNum = numCheckpoints[dstRep];
890             uint256 dstRepOld = dstRepNum > 0 ? checkpoints[dstRep][dstRepNum -
891             1].votes : 0;
892             uint256 dstRepNew = dstRepOld.add(amount);
893             _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
894         }
895     }
896 }
```

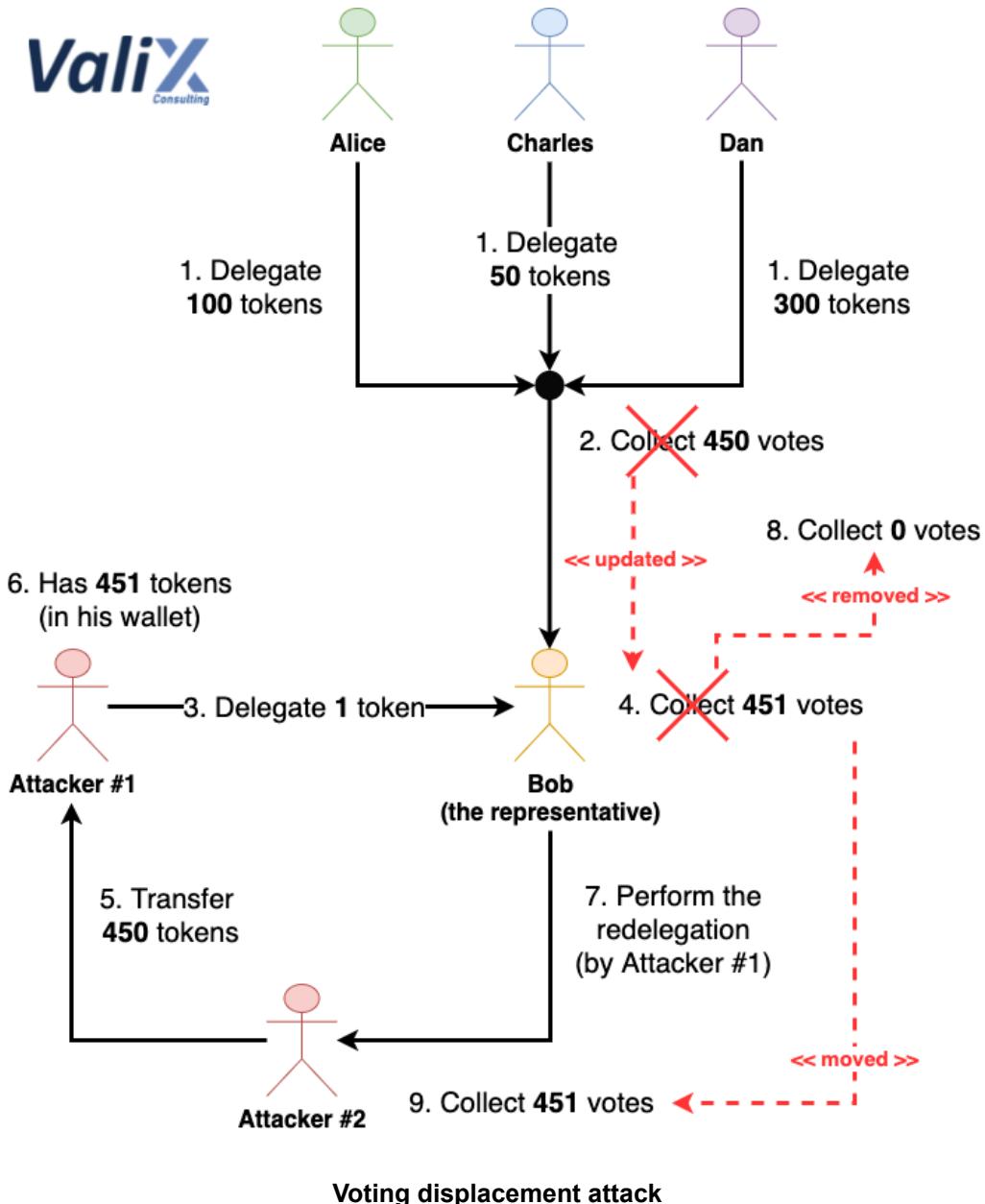
The code snippet above shows the `_delegate` and `_moveDelegates` functions that are the root cause of the issue. During the redelegation process, the `_delegate` function would be executed. This function gets the delegator's current representative (line no. 865). Then, the function reads the delegator's current VON balance (line no. 866). Next, the function changes the representative to the new one (line no. 867).

The delegator's VON balance (from line no. 866) is then passed into the `_moveDelegates` function (line no. 871) and becomes the function parameter, `amount`.

In the `_moveDelegates` function, the old representative's votes are decreased by the variable amount (line no. 880). The exact amount is also increased to the new representative's votes (line no. 888). In other words, the votes will be moved from the old to the new representative.

Since the amount of the moved votes is determined by the delegator's current VON balance, not the previously delegated VONs, the attackers can manipulate the incorrect number of the votes movement.

To conclude, the delegator's VON balance (line no. 866) is the root cause of the issue.



Consider the voting displacement scenario illustrated in the figure above.

1. **Alice, Charles, and Dan** delegate 100, 50, and 300 tokens respectively to **Bob**
2. **Bob** collects 450 votes
3. **Attacker #1** initially has 1 token and delegates his vote to **Bob**
4. **Bob** collects 451 votes for now
5. **Attacker #2** transfers his 450 tokens to **Attacker #1**
6. **Attacker #1** now has 451 tokens in his wallet
7. **Attacker #1** re-delegates his vote to **Attacker #2**
8. **Bob's** collected votes are improperly removed by 451 (i.e., the current token balance of **Attacker #1**) and finally become 0
9. **Attacker #2** eventually receives the *manipulated* 451 votes

The **450 votes** (delegated by **Alice, Charles, and Dan**) to **Bob** are improperly removed at Step 8 due to the design flaw explained earlier. Hence, **the attackers can use this voting displacement attack to dismiss the votes of other voters easily.**

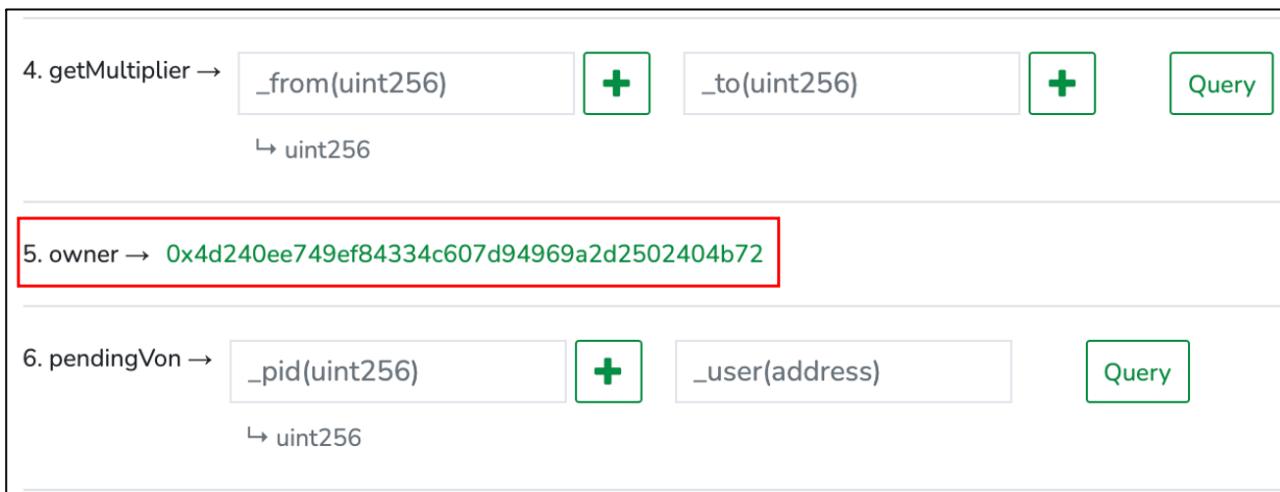
Reassessment

The affected voting functionality was removed from the VonderToken contract.

No. 3	Contract Parameters Can Be Altered By The Platform Developer Without Timelock				
Risk	High	Likelihood	Medium		
Functionality is in use	In use	Impact	High		
Affected Files	MasterChef.sol	Status	Partially Fixed		
Locations	-				
Description					
<p>The MasterChef contract is owned by an Externally Owned Account (EOA) without having an intermediate contract to prevent an administrative user from executing an arbitrary function immediately. The state variables such as <code>vonPerBlock</code> (emission rate) could be updated and effective immediately without user notification.</p>					
Recommendations					
<p>Implement the Timelock contract and transfer the ownership of the MasterChef contract to the Timelock contract.</p> <p>The linkage of the associated contracts should be as follows:</p> <p>Deployer or multi-signature contract --> Timelock --> MasterChef</p> <p>Specifically, the MasterChef is owned by the Timelock, whereas the Timelock is owned by a deployer (admin).</p> <p>It is acceptable to use an externally owned account as an administrator of the Timelock. For best practices, a multi-signature contract should be an administrator of the Timelock.</p> <p>Reference: https://docs.gnosis.io/safe/docs/contracts_architecture/</p>					
Platform Developer Response					
<p>The developer implemented the Timelock contract for resolving this issue.</p>					

Detailed Issue

According to the MasterChef contract on the BKCScan (<https://bkcsan.com/address/0x60326f6Ad05adeE2ffD42B0c05c68Ead535B104E>), its owner address was 0x4d240ee749ef84334c607d94969a2d2502404b72.



4. getMultiplier → `_from(uint256)`  `_to(uint256)`  

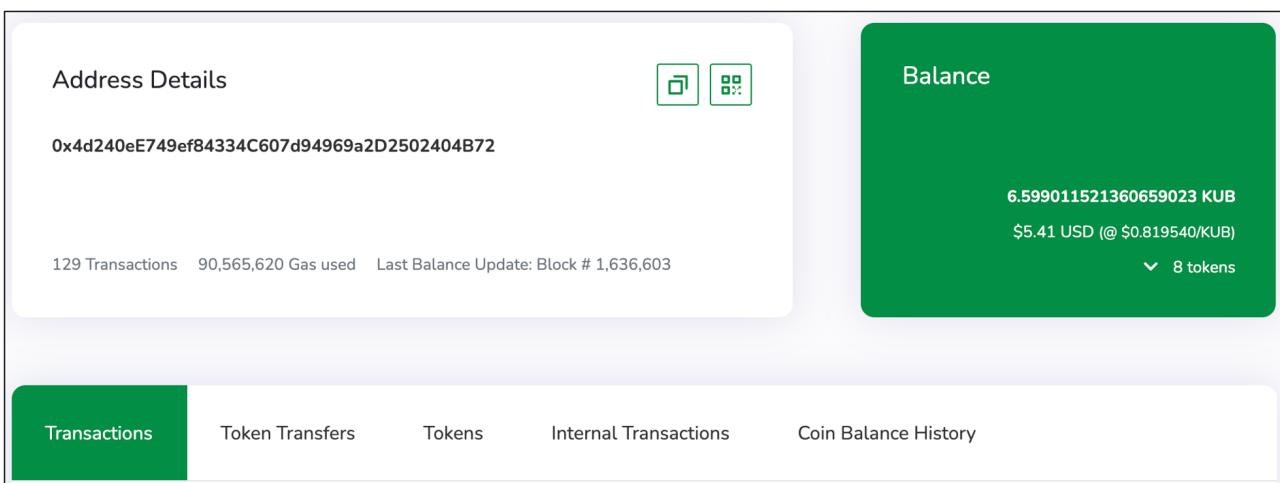
↳ uint256

5. owner → `0x4d240ee749ef84334c607d94969a2d2502404b72`

6. pendingVon → `_pid(uint256)`  `_user(address)` 

↳ uint256

Apparently, the owner of the MasterChef contract was an Externally Owned Account (EOA) wallet:
<https://bkcsan.com/address/0x4d240eE749ef84334C607d94969a2D2502404B72>.



Address Details

0x4d240eE749ef84334C607d94969a2D2502404B72

129 Transactions 90,565,620 Gas used Last Balance Update: Block # 1,636,603

Balance

6.599011521360659023 KUB

\$5.41 USD (@ \$0.819540/KUB)

▼ 8 tokens

Transactions

Token Transfers

Tokens

Internal Transactions

Coin Balance History

Since the EOA account (admin) can immediately change and affect the platform's parameters, users cannot have time to inspect any parameter changes.

For example, the state variable `vonPerBlock` could be changed and take effect immediately.

MasterChef.sol

```
464 function updateEmissionRate(uint256 _vonPerBlock) public onlyOwner {
465     massUpdatePools();
466     vonPerBlock = _vonPerBlock;
467 }
```

Reassessment

The developer implemented the Timelock contract for resolving this issue.

Timelock.sol

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity 0.6.12;
4
5 import './libs/SafeMath.sol';
6
7 contract Timelock {
8     using SafeMath for uint;
9
10    event NewAdmin(address indexed newAdmin);
11    event NewPendingAdmin(address indexed newPendingAdmin);
12    event NewDelay(uint indexed newDelay);
13    event CancelTransaction(bytes32 indexed txHash, address indexed target, uint
value, string signature, bytes data, uint eta);
14    event ExecuteTransaction(bytes32 indexed txHash, address indexed target, uint
value, string signature, bytes data, uint eta);
15    event QueueTransaction(bytes32 indexed txHash, address indexed target, uint
value, string signature, bytes data, uint eta);
16
17    uint public constant GRACE_PERIOD = 14 days;
18    uint public constant MINIMUM_DELAY = 6 hours;
19    uint public constant MAXIMUM_DELAY = 30 days;
...
}
```

However, we found that the `setPendingAdmin` function of the Timelock contract allows the developer to set the state variable `pendingAdmin` without time delay (line no's. 63 - 66) for the first call of the `admin` address changes. In other words, the developer can change the `admin` address for the first time immediately.

Timelock.sol

```
58  function setPendingAdmin(address pendingAdmin_) public {
59      // allows one time setting of admin for deployment purposes
60      if (admin_initialized) {
61          require(msg.sender == address(this), "Timelock::setPendingAdmin: Call
must come from Timelock.");
62      } else {
63          require(msg.sender == admin, "Timelock::setPendingAdmin: First call must
come from admin.");
64          admin_initialized = true;
65      }
66      pendingAdmin = pendingAdmin_;
67
68      emit NewPendingAdmin(pendingAdmin);
69  }
```

We notified this concern to the Vonder team. The team acknowledged our concern but decided to make no further improvements.

No. 4	Redelegation Failure					
Risk	High	Likelihood	High			
Functionality is in use	Not in use	Impact	Medium			
Affected Files	<i>VonderToken.sol</i>					
Locations	<code>_delegate(address, address) L:862 - 872</code> <code>_moveDelegates(address, address, uint256) L:874 - 892</code>					
Description						
<p>A voter/delegator (VON token holder) can re-delegate his votes to another representative by calling the external <code>delegate</code> or <code>delegateBySig</code> function. The external function will subsequently call the internal <code>_delegate</code> function which will obtain a delegator balance.</p> <p>The <code>_moveDelegates</code> function is then invoked to move the delegator's votes (the previously obtained delegator balance) from the current to the new representative.</p> <p>There are some situations where the voter/delegator cannot re-delegate their votes. Consider the following redelegation scenario:</p> <ol style="list-style-type: none"> 1. Alice has <i>100 tokens</i> and delegates her vote to Bob. 2. Alice receives additional <i>10 tokens</i> from her yield farming. 3. If Alice attempts to re-delegate her <i>110 tokens</i> to Dan, the transaction will fail since the <code>_moveDelegates</code> function will try to un-delegate <i>110</i> (not <i>100</i>) votes from Bob, causing the <i>sub</i> function of the <i>SafeMath</i> library to revert. 						
Recommendations						
<p>We recommend two possible solutions. The first solution is improving the <i>VonderToken</i> contract to lock away the delegated VON tokens inside until the voting or delegating period is complete. The <i>VonderToken</i> contract also has to record the number of votes of each delegator correctly so that the contract can check and move each delegator's votes precisely when re-delegating.</p> <p>Another solution is implementing another <i>voting</i> contract and using VON tokens as the contract's voting tokens. The <i>voting</i> contract also needs to lock up and record the delegated VON tokens correctly nonetheless.</p>						
Platform Developer Response						
<p>The affected voting functionality was removed from the <i>VonderToken</i> contract.</p>						

Detailed Issue

This issue causes a transaction revert during the redelegation process, which can affect every regular voter.

VonderToken.sol

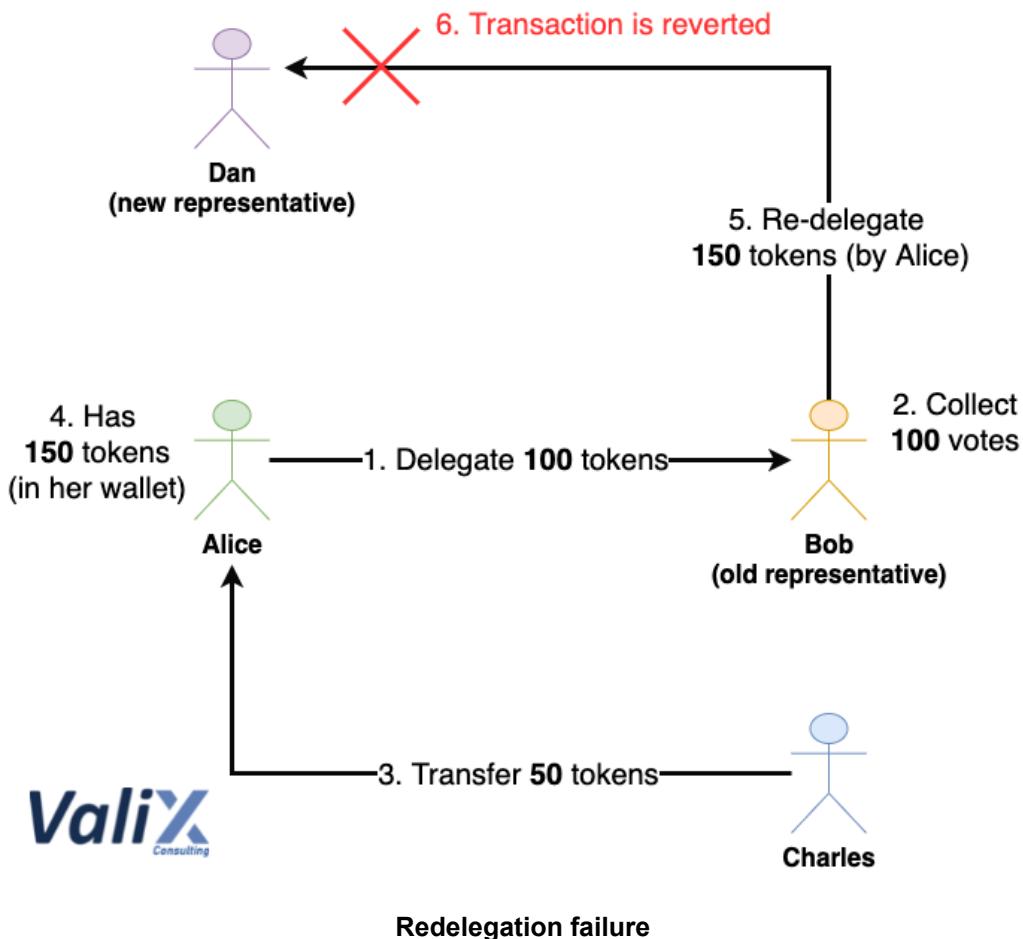
```

862 function _delegate(address delegator, address delegatee)
863     internal
864 {
865     address currentDelegate = _delegates[delegator];
866     uint256 delegatorBalance = balanceOf(delegator); // balance of underlying
867     VONs (not scaled);
868     _delegates[delegator] = delegatee;
869
870     emit DelegateChanged(delegator, currentDelegate, delegatee);
871
872     _moveDelegates(currentDelegate, delegatee, delegatorBalance);
873 }
874
875 function _moveDelegates(address srcRep, address dstRep, uint256 amount) internal
876 {
877     if (srcRep != dstRep && amount > 0) {
878         if (srcRep != address(0)) {
879             // decrease old representative
880             uint32 srcRepNum = numCheckpoints[srcRep];
881             uint256 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum - 1].votes : 0;
882             uint256 srcRepNew = srcRepOld.sub(amount);
883             _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
884         }
885         if (dstRep != address(0)) {
886             // increase new representative
887             uint32 dstRepNum = numCheckpoints[dstRep];
888             uint256 dstRepOld = dstRepNum > 0 ? checkpoints[dstRep][dstRepNum - 1].votes : 0;
889             uint256 dstRepNew = dstRepOld.add(amount);
890             _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
891         }
892     }
893 }
```

The code snippet above points out the root cause of the issue; the `_delegate` and `_moveDelegates` functions. The `_moveDelegates` function will move a certain amount of votes from the old (line no's. 878–881) to the new representative (line no's. 886–889). The votes movement amount is determined by the delegator's current VON balance (line no. 866) in the `_delegate` function.

During the re-delegation process, the transaction would be reverted in line no. 880 if the delegator has more VON balance than the votes previously recorded.

More specifically, the `_moveDelegates` function would attempt to deduct the surpassing number from the exact number recorded, causing an **integer underflow** error. Thus, the `sub` function of the `SafeMath` library would revert the transaction.



The redelegation failure scenario can be depicted using the figure above.

1. Alice initially has *100 tokens* and delegates her vote to **Bob**
2. **Bob** obtains *100 votes* now
3. **Charles** transfers his *50 tokens* to **Alice**
4. **Alice** now has *150 tokens* in her wallet
5. **Alice** tries to re-delegate her votes to another representative, **Dan**
6. The **redelegation transaction** is reverted due to the **integer underflow** error

Three possible actions can cause Alice's transaction to revert.

1. Alice receives additional tokens from the **token transfer** (from others)
2. Alice receives additional tokens from the **token buying**
3. Alice receives additional tokens from the **yield farming**

This issue can affect both the **voting redelegation** and **voting withdrawal** transactions invoked by a regular voter.

Reassessment

The affected voting functionality was removed from the VonderToken contract.

No. 5	No Maximum Supply Minting Check					
Risk	Medium	Likelihood	Medium			
Functionality is in use	In use	Status	Fixed			
Affected Files	<i>VonderToken.sol</i>					
Locations	<i>mint(address, uint256) L: 690-693</i> <i>_mint(address, uint256) L: 537-543</i>					
Description						
<p>According to Vonder's official documentation, the VON maximum supply is 101,051,200. The MasterChef contract is responsible for minting the new VON tokens to distribute as a reward to the users staking the liquidity pools.</p> <p>However, the Vonder developer did not expressly declare the VON maximum supply in the VonderToken contract. Over time, the MasterChef contract can mint excessive tokens.</p> <p>Reference: https://docs.vonder.finance/tokenomics-1/tokenomics/vonder-emission-schedule</p>						
Recommendations						
<p>Implement the statement to check whether the <i>totalSupply</i> is more than the maximum supply or not. The statement should be checked before minting the new VON token. For example, consider the pseudo-code below:</p> <pre>uint private _maxSupply = 101051200e18; function mint(address _to, uint256 _amount) public onlyOwner { require(totalSupply().add(_amount) <= _maxSupply, "VON exceeds maxSupply"); _mint(_to, _amount); _moveDelegates(address(0), _delegates[_to], _amount); }</pre>						
Platform Developer Response						
<p>The developer implemented the maximum supply check to resolve this issue.</p>						

Detailed Issue

On the `updatePool` function, new VON tokens will be minted to the `devaddr` and the `MasterChef` contract itself by calling the `mint` function of the `VonderToken` contract.

MasterChef.sol

```

370 function updatePool(uint256 _pid) public {
371     PoolInfo storage pool = poolInfo[_pid];
372     if (block.number <= pool.lastRewardBlock) {
373         return;
374     }
375     uint256 lpSupply = pool.lpToken.balanceOf(address(this));
376     if (lpSupply == 0 || pool.allocPoint == 0) {
377         pool.lastRewardBlock = block.number;
378         return;
379     }
380     uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
381     uint256 vonReward =
382         multiplier.mul(vonPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
383         von.mint(devaddr, vonReward.div(10));
384         von.mint(address(this), vonReward);
385     pool.accVonPerShare =
386         pool.accVonPerShare.add(vonReward.mul(1e12).div(lpSupply));
387     pool.lastRewardBlock = block.number;
388 }
```

The `mint` function of the `VonderToken` contract calls the internal `_mint` function.

VonderToken.sol

```

500 function mint(uint256 amount) public onlyOwner returns (bool) {
501     _mint(_msgSender(), amount);
502     return true;
503 }
```

According to Vonder's official documentation, the VON maximum supply is 101,051,200. In the `_mint` function, however, there is no maximum supply checking. Therefore, the MasterChef contract can mint excessive tokens over time.

VonderToken.sol

```

537     function _mint(address account, uint256 amount) internal {
538         require(account != address(0), 'BEP20: mint to the zero address');
539
540         _totalSupply = _totalSupply.add(amount);
541         _balances[account] = _balances[account].add(amount);
542         emit Transfer(address(0), account, amount);
543     }

```

Reassessment

The maximum supply is checked in the `mint` function of the VonderToken contract.

VonderToken.sol

```

547     contract VonderToken is BEP20('Extended VONDER Token', 'xVON') {
548         uint256 private _cap = 101051200e18; //101,051,200
549
550         function cap() public view returns (uint256) {
551             return _cap;
552         }
553
554         // @notice Creates `_amount` token to `_to`. Must only be called by the owner
555         // (MasterChef).
556         function mint(address _to, uint256 _amount) public onlyOwner {
557             require(totalSupply().add(_amount) <= cap(), "cap exceeded");
558             _mint(_to, _amount);
559             // _moveDelegates(address(0), _delegates[_to], _amount);
560         }
561     }

```

No. 6	No LP Token Adding After Deposit					
Risk	Medium	Likelihood	Low			
	Impact	High				
Functionality is in use	In use	Status	Fixed			
Affected Files	<i>MasterChef.sol</i>					
Locations	<i>deposit(uint256, uint256) L:389-411</i>					
Description						
On the <i>deposit</i> function, if the state variable <i>pool.depositFeeBP</i> was set to <i>10000</i> , the computed <i>depositFee</i> variable will equal the user's <i>deposit_amount</i> (line no. 402). The total deposit amount will be transferred to the fee address as a deposit fee (line no. 403). Thus, no LP token will be left adding to the user account (line no. 404).						
Recommendations						
Limit a maximum cap for the <i>depositFeeBP</i> variable to less than <i>10000</i> . For example, if the maximum cap is 5000. Therefore, the maximum deposit fee is 50%.						
Platform Developer Response						
The developer set the maximum cap of the <i>depositFeeBP</i> variable to 5000. That is, the maximum deposit fee is 50%.						

Detailed Issue

On the `deposit` function, if the state variable `pool.depositFeeBP` was set to `10000`, the computed `depositFee` variable will equal the user's deposit `_amount` (line no. 402). The total deposit amount will be transferred to the fee address as a deposit fee (line no. 403). Thus, no LP token will be left adding to the user account (line no. 404).

MasterChef.sol

```

389  function deposit(uint256 _pid, uint256 _amount) public {
390      PoolInfo storage pool = poolInfo[_pid];
391      UserInfo storage user = userInfo[_pid][msg.sender];
392      updatePool(_pid);
393      if (user.amount > 0) {
394          uint256 pending =
395              user.amount.mul(pool.accVonPerShare).div(1e12).sub(user.rewardDebt);
396          if(pending > 0) {
397              safeVonTransfer(msg.sender, pending);
398          }
399          if(_amount > 0) {
400              pool.lpToken.safeTransferFrom(address(msg.sender), address(this),
401                  _amount);
402              if(pool.depositFeeBP > 0){
403                  uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
404                  pool.lpToken.safeTransfer(feeAddress, depositFee);
405                  user.amount = user.amount.add(_amount).sub(depositFee);
406              }else{
407                  user.amount = user.amount.add(_amount);
408              }
409          user.rewardDebt = user.amount.mul(pool.accVonPerShare).div(1e12);
410          emit Deposit(msg.sender, _pid, _amount);
411      }

```

Reassessment

The issue was fixed by limiting the maximum cap of the `depositFeeBP` variable to `5000` in the `add` and `set` functions of the `MasterChef` contract. The maximum deposit fee, therefore, became 50%.

MasterChef.sol

```

315  function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, bool
316  _withUpdate) public onlyOwner {
317      require(_depositFeeBP <= 5000, "add: invalid deposit fee basis points");
318      if (_withUpdate) {
319          massUpdatePools();
320      }
321      uint256 lastRewardBlock = block.number > startBlock ? block.number :
322      startBlock;
323      totalAllocPoint = totalAllocPoint.add(_allocPoint);
324      poolInfo.push(PoolInfo({
325          lpToken: _lpToken,
326          allocPoint: _allocPoint,
327          lastRewardBlock: lastRewardBlock,
328          accVonPerShare: 0,
329          depositFeeBP: _depositFeeBP
330      }));
331  }
332  // Update the given pool's VON allocation point and deposit fee. Can only be
333  // called by the owner.
334  function set(uint256 _pid, uint256 _allocPoint, uint16 _depositFeeBP, bool
335  _withUpdate) public onlyOwner {
336      require(_depositFeeBP <= 5000, "set: invalid deposit fee basis points");
337      if (_withUpdate) {
338          massUpdatePools();
339      }
340      totalAllocPoint =
341      totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
342      poolInfo[_pid].allocPoint = _allocPoint;
343      poolInfo[_pid].depositFeeBP = _depositFeeBP;
344  }

```

No. 7	Emission Rate Update May Fail				
Risk	Low	Likelihood	Low		
Functionality is in use	In use	Impact	Medium		
Affected Files	<i>MasterChef.sol</i>	Status	Acknowledged		
Locations	<i>updateEmissionRate(uint256) L:464-467</i>				
Description					
<p>The <i>updateEmissionRate</i> function calls the <i>massUpdatePools</i> function before updating the emission rate variable <i>vonPerBlock</i>. Since the <i>massUpdatePools</i> function will iterate over all the pools to update pools' reward variables, this function consumes as much gas as the number of pools in the Vonder system.</p> <p>If the Vonder system has more pools, the <i>massUpdatePools</i> function may consume more gas than the gas limit per block, causing a transaction failure with an out-of-gas error. Consequently, the <i>updateEmissionRate</i> function will not be able to update the <i>vonPerBlock</i> variable anymore.</p>					
Recommendations					
<p>The first solution is adding the boolean parameter <i>_withUpdate</i> to the <i>updateEmissionRate</i> function to enable a platform owner to update the <i>vonPerBlock</i> variable without the mass pools update.</p> <p>Another solution is separating the tasks between the mass pools update and the <i>vonPerBlock</i> variable update into different functions.</p>					
Platform Developer Response					
<p>The Vonder team acknowledged this issue but decided to remain the original code.</p>					

Detailed Issue

The `updateEmissionRate` function calls the `massUpdatePools` function (line no. 465) before updating the emission rate variable `vonPerBlock` (line no. 466). Since the `massUpdatePools` function will iterate over all the pools to update pools' reward variables, this function consumes as much gas as the number of pools in the Vonder system.

If the Vonder system has more pools, the `massUpdatePools` function may consume more gas than the gas limit per block, causing a transaction failure with an out-of-gas error. Consequently, the `updateEmissionRate` function will not be able to update the `vonPerBlock` variable anymore.

Masterchef.sol

```
464     function updateEmissionRate(uint256 _vonPerBlock) public onlyOwner {  
465         massUpdatePools();  
466         vonPerBlock = _vonPerBlock;  
467     }
```

Reassessment

The Vonder team acknowledged this issue but decided to remain the original code.

No. 8	<i>DevAddress Reassignment May Fail</i>					
Risk	Low	Likelihood	Low			
	Impact	Medium				
Functionality is in use	In use	Status	Acknowledged			
Affected Files	<i>MasterChef.sol</i>					
Locations	<i>dev(address) L:453 - 456</i>					
Description						
<p>The address variable <code>_devaddr</code> on the <code>dev</code> function may be incorrectly specified by the platform developer by mistake; for example, a zero address or an address that the developer does not own was inputted.</p> <p>The incorrectly inputted address makes the <code>dev</code> function unavailable since it strictly checks that the user who can change the new address must be the current <code>devaddr</code> only. But if the platform developer cannot access the mistakenly inputted address, he cannot change the <code>devaddr</code> anymore because of the <code>require</code> statement in line no. 454.</p>						
Recommendations						
<p>To prevent human error, we recommend allowing another address (e.g., the contract's owner address) to be able to execute the <code>dev</code> function as a backup account.</p> <p>We recommend allowing another address (e.g., the contract's owner address) to execute the <code>dev</code> function as a backup account to protect against human error.</p>						
Platform Developer Response						
<p>The Vonder team acknowledged this issue but decided to remain the original code.</p>						

Detailed Issue

The address variable `_devaddr` on the `dev` function may be incorrectly specified by the platform developer by mistake; for example, a zero address or an address that the developer does not own was inputted.

The incorrectly inputted address makes the `dev` function unavailable since it strictly checks that the user who can change the new address must be the current `devaddr` only. But if the platform developer cannot access the mistakenly inputted address, he cannot change the `devaddr` anymore because of the `require` statement in line no. 454.

Masterchef.sol

```
453 function dev(address _devaddr) public {
454     require(msg.sender == devaddr, "dev: wut?");
455     devaddr = _devaddr;
456 }
```

Reassessment

The Vonder team acknowledged this issue but decided to remain the original code.

No. 9	FeeAddress Reassignment May Fail					
Risk	Low	Likelihood	Low			
	Impact	Medium				
Functionality is in use	In use	Status	Acknowledged			
Affected Files	<i>MasterChef.sol</i>					
Locations	<i>setFeeAddress(address) L:458 - 461</i>					
Description						
<p>The address variable <code>_feeAddress</code> on the <code>setFeeAddress</code> function may be incorrectly specified by the platform developer by mistake; for example, a zero address or an address that the developer does not own was inputted.</p> <p>The incorrectly inputted address makes the <code>setFeeAddress</code> function unavailable since it strictly checks that the user who can change the new address must be the current <code>feeAddress</code> only. But if the platform developer cannot access the mistakenly inputted address, he cannot change the <code>feeAddress</code> anymore because of the <code>require</code> statement in line no. 459.</p>						
Recommendations						
<p>We recommend allowing another address (e.g., the contract's owner address) to execute the <code>setFeeAddress</code> function as a backup account to protect against human error.</p>						
Platform Developer Response						
<p>The Vonder team acknowledged this issue but decided to remain the original code.</p>						

Detailed Issue

The address variable `_feeAddress` on the `setFeeAddress` function may be incorrectly specified by the platform developer by mistake; for example, a zero address or an address that the developer does not own was inputted.

The incorrectly inputted address makes the `setFeeAddress` function unavailable since it strictly checks that the user who can change the new address must be the current `feeAddress` only. But if the platform developer cannot access the mistakenly inputted address, he cannot change the `feeAddress` anymore because of the `require` statement in line no. 459.

Masterchef.sol

```
458     function setFeeAddress(address _feeAddress) public{
459         require(msg.sender == feeAddress, "setFeeAddress: FORBIDDEN");
460         feeAddress = _feeAddress;
461     }
```

Reassessment

The Vonder team acknowledged this issue but decided to remain the original code.

No. 10	The Compiler May Be Susceptible To The Publicly Disclosed Bugs					
Risk	Low	Likelihood	Low			
	Impact	Medium				
Functionality is in use	In use	Status	Acknowledged			
Affected Files	<i>MasterChef.sol</i> <i>VonderToken.sol</i>					
Locations	<i>MasterChef.sol L:3</i> <i>VonderToken.sol L:2</i>					
Description						
The contract uses an outdated Solidity compiler version which may be susceptible to publicly disclosed vulnerabilities. The compiler version currently used by Vonder Finance is 0.6.6 which contains the list of known bugs as the following link: https://docs.soliditylang.org/en/v0.6.6/bugs.html The known bugs may not directly lead to the vulnerability, but it may increase an opportunity to trigger some attacks further.						
Recommendations						
We recommend using the latest patch version, v0.6.12.						
Platform Developer Response						
The Vonder team acknowledged this issue but decided to remain the original code.						

Detailed Issue

The usage example of the Solidity compiler is not the latest patch version (v0.6.12).

Masterchef.sol

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.6.6;
4
5 import "./VonderToken.sol";
6
7 library Address {
```

Reassessment

The Vonder team acknowledged this issue but decided to remain the original code.

No. 11	The Compiler Is Not Locked To A Specific Version					
Risk	Low	Likelihood	Low			
	Impact	Medium				
Functionality is in use	In use	Status	Acknowledged			
Affected Files	<i>MasterChef.sol</i> <i>VonderToken.sol</i>					
Locations	<i>MasterChef.sol L:3</i> <i>VonderToken.sol L:2</i>					
Description						
Contract should be deployed with the compiler version that is used in a development and testing process. The compiler version that is not strictly locked via the <i>pragma</i> statement leads the contract to be incompatible against unforeseen circumstances.						
Recommendations						
Lock the pragma version like the example code snippet below.						
<pre>pragma solidity 0.8.6; // or pragma solidity =0.8.6; contract SemVerFloatingPragmaFixed {</pre>						
Reference: https://swcregistry.io/docs/SWC-103						
Platform Developer Response						
The Vonder team acknowledged this issue but decided to remain the original code.						

Detailed Issue

The example of the Solidity compiler that is not locked to a specific version (i.e., using `>=` or `^` directive).

Masterchef.sol

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.6.6;
4
5 import "./VonderToken.sol";
6
7 library Address {
```

Reassessment

The Vonder team acknowledged this issue but decided to remain the original code.

No. 12	Same LP Token May Be Added More Than Once					
Risk	Informational	Likelihood	Low			
		Impact	Low			
Functionality is in use	In use	Status	Acknowledged			
Affected Files	<i>MasterChef.sol</i>					
Locations	<i>add(uint256, IBEP20, uint16, bool) L:315-329</i>					
Description						
The <i>add</i> function allows a platform developer to add the same LP token to the yield farming system without verifying that the token has previously been added. If the same LP token is added more than once, this will affect the reward distribution parameters, such as <i>totalAllLocPoint</i> , as well as affecting the user experience.						
Recommendations						
Verify the duplication of the LP token before adding it to the yield farming system.						
Platform Developer Response						
The Vonder team acknowledged this issue but decided to remain the original code.						

Detailed Issue

Masterchef.sol

```

315   function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, bool
316   _withUpdate) public onlyOwner {
317     require(_depositFeeBP <= 10000, "add: invalid deposit fee basis points");
318     if (_withUpdate) {
319       massUpdatePools();
320     }
321     uint256 lastRewardBlock = block.number > startBlock ? block.number :
322     startBlock;
323     totalAllocPoint = totalAllocPoint.add(_allocPoint);
324     poolInfo.push(PoolInfo({
325       lpToken: _lpToken,
326       allocPoint: _allocPoint,
327       lastRewardBlock: lastRewardBlock,
328       accVonPerShare: 0,
329       depositFeeBP: _depositFeeBP
330     }));
331   }

```

The `add` function allows a platform developer to add the same LP token to the yield farming system without verifying that the token has previously been added. If the same LP token is added more than once, this will affect the reward distribution parameters, such as `totalAllocPoint`, as well as affecting the user experience.

Reassessment

The Vonder team acknowledged this issue but decided to remain the original code.

No. 13	The Function Name With <i>internal</i> Visibility Is Not Complied With The Naming Convention					
Risk	Informational	Likelihood	Low			
		Impact	Low			
Functionality is in use	In use	Status	Acknowledged			
Affected Files	<i>MasterChef.sol</i>					
Locations	<i>safeVonTransfer(address, uint256)</i> L:443 - 450					
Description						
The coding style in the contract is inconsistent due to an incompliant Solidity style guide leading to a code transfer disadvantage, or loss of backward compatibility.						
Recommendations						
The internal or private variables should be used "_" at the beginning.						
Platform Developer Response						
The Vonder team acknowledged this issue but decided to remain the original code.						

Detailed Issue

The internal function does not comply with the Solidity Style guide.

Masterchef.sol
<pre> 443 function safeVonTransfer(address _to, uint256 _amount) internal { 444 uint256 vonBal = von.balanceOf(address(this)); 445 if (_amount > vonBal) { 446 von.transferWithLock(_to, vonBal); 447 } else { 448 von.transferWithLock(_to, _amount); 449 } 450 }</pre>

Reassessment

The Vonder team acknowledged this issue but decided to remain the original code.

No. 14	Public Functions That Could Be Declared As <i>external</i>				
Risk	Informational	Likelihood	Low		
		Impact	Low		
Functionality is in use	In use	Status	Acknowledged		
Affected Files	<i>Masterchef.sol</i> <i>VonderToken.sol</i>				
Locations					
<i>Masterchef.sol</i> <i>add(uint256, IBEP20, uint16, bool) L:315</i> <i>set(uint256, uint256, uint16, bool) L:332</i> <i>deposit(uint256, uint256) L:389</i> <i>withdraw(uint256, uint256) L:414</i> <i>emergencyWithdraw(uint256) L:432</i> <i>dev(address) L:453</i> <i>setFeeAddress(address) L:458</i> <i>updateEmissionRate(uint256) L:464</i> <i>setVonRewardLock(uint256) L:470</i> <i>setVonTotalBlockRelease(uint256) L:474</i>		<i>VonderToken.sol (Cont'd)</i> <i>totalSupply() L:391</i> <i>transfer(address, uint256) L:410</i> <i>allowance(address, address) L:418</i> <i>approve(address, uint256) L:427</i> <i>transferFrom (address, address, uint256) L:446</i> <i>increaseAllowance(address, uint256) L:468</i> <i>decreaseAllowance(address, uint256) L:487</i> <i>mint(uint256) L:500</i> <i>setRewardLock(uint256) L:605</i> <i>setTotalBlockRelease(uint256) L:611</i> <i>transferWithLock(address, uint256) L:616</i> <i>claimRewardLock() L:645</i> <i>getTotalRewardLock(address) L:674</i> <i>getLastClaimBlock(address) L:678</i> <i>getEndClaimBlock(address) L:682</i> <i>mint(address, uint256) L: 690</i>			
Description					
The <i>public</i> functions that have never been called inside the contracts should be declared <i>external</i> to save gas.					
Recommendations					
Use the <i>external</i> attribute for functions that have never been called inside the contracts.					
Platform Developer Response					
The Vonder team acknowledged this issue but decided to remain the original code.					

Detailed Issue

An example of the *public* function that has never been called inside any contract but not declared *external*.

Masterchef.sol

```
315  function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, bool
316    _withUpdate) public onlyOwner {
317      require(_depositFeeBP <= 10000, "add: invalid deposit fee basis points");
318      if (_withUpdate) {
319        massUpdatePools();
320      }
321      uint256 lastRewardBlock = block.number > startBlock ? block.number :
322      startBlock;
323      totalAllocPoint = totalAllocPoint.add(_allocPoint);
324      poolInfo.push(PoolInfo({
325        lpToken: _lpToken,
326        allocPoint: _allocPoint,
327        lastRewardBlock: lastRewardBlock,
328        accVonPerShare: 0,
329        depositFeeBP: _depositFeeBP
330      }));
331    }
```

Reassessment

The Vonder team acknowledged this issue but decided to remain the original code.

Appendix

About Us

Founded in 2020, Valix Consulting is a blockchain and smart contract security firm offering a wide range of cybersecurity consulting services such as blockchain and smart contract security consultant, smart contract security review, and smart contract security audit.

Our team members are passionate cybersecurity professionals and researchers in areas of private and public blockchain technology, smart contract, and decentralized application (DApp).

We provide a service for assessing and certifying the security of smart contracts. Our service also includes recommendations on smart contracts' security and gas optimization to bring the most benefit to users and platform creators.

Contact Information



info@valix.io



<https://www.facebook.com/ValixConsulting>



<https://twitter.com/ConsultingValix>



<https://medium.com/valixconsulting>

References

Title	Link
OWASP Risk Rating Methodology	https://owasp.org/www-community/OWASP_Risk_Rating_Methodology
Smart Contract Weakness Classification and Test Cases	https://swcregistry.io/



Valix