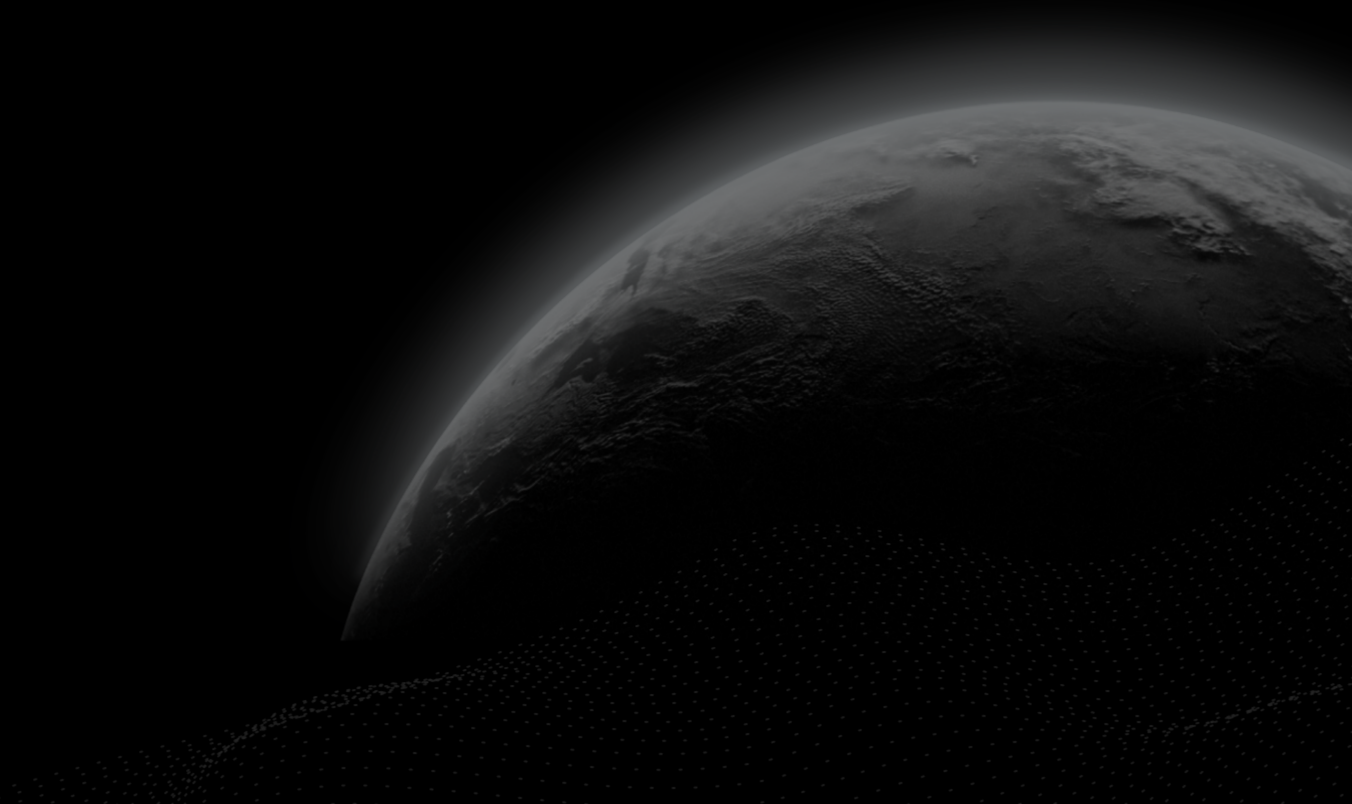
CERTIK

Security Assessment

WEMIX Staking (GRAND Staking, DIOS Staking)

CertiK Verified on Oct 13th, 2022



CertiK Verified on Oct 13th, 2022

WEMIX Staking (GRAND Staking, DIOS Staking)

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Staking	Ethereum	Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 10/13/2022	N/A

CODEBASE

https://github.com/wemixarchive/WemixFi-Staking/

...View All

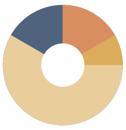
COMMITS

base: d9222d9933e4dc56b68ec542d714342024ceb97b

update: fd4752c578dedfced8d5bf49aac7b14e945e4d2d

...View All

Vulnerability Summary



12	7	0	1	4	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

🟥	0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟧	2	Major	2 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
🟨	1	Medium	1 Partially Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨	7	Minor	5 Resolved, 2 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦	2	Informational	2 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | WEMIX STAKING (GRAND STAKING, DIOS STAKING)

CODEBASE | WEMIX STAKING (GRAND STAKING, DIOS STAKING)

Repository

https://github.com/wemixarchive/WemixFi-Staking/

Commit

base: d9222d9933e4dc56b68ec542d714342024ceb97b

update: fd4752c578dedfced8d5bf49aac7b14e945e4d2d

AUDIT SCOPE | WEMIX STAKING (GRAND STAKING, DIOS STAKING)

7 files audited ● 1 file with Acknowledged findings ● 1 file with Resolved findings ● 5 files without findings

ID	File	SHA256 Checksum
● SWC	Staking.sol	943f632d86c9992c094c5e2d333a6585c28c4d744c33015cd86b0be9e9f57a73
● RWC	Rewarder.sol	f07e0ba6e893bbe080380fefd5c8a4f9bcefc810462f583a9fb50507741413d4
● IES	interfaces/IEnvStorage.sol	68ef23e7138c7e886eca98fd729af807e289ea2cd0765048967696351020e17f
● IRW	interfaces/IRewarder.sol	e8f1181d286e39be9461ff2b2442b2ca62d63112590d7be569f1b06800169633
● ISW	interfaces/IStaking.sol	464bc2782903af1972852f75b15b5dc9a411acfe25f3443e893e644e36dc87e5
● IWW	interfaces/IWWEMIX.sol	711f8fdd6f1445468708a5ea3470a42c2f3e8bce61df17932c76031550602333
● IWR	interfaces/IWeswapRouter.sol	5e2fafd76ede543521f6123be2079ab36e9d569350698d43b3c8d50cefa2df68

APPROACH & METHODS | WEMIX STAKING (GRAND STAKING, DIOS STAKING)

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

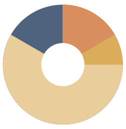
The auditing process pays special attention to the following considerations:

- 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:

- Testing the smart contracts against both common and uncommon attack vectors;
- 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.

FINDINGS | WEMIX STAKING (GRAND STAKING, DIOS STAKING)



12	0	2	1	7	2
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for WEMIX Staking (GRAND Staking, DIOS Staking) . Through this audit, we have uncovered 12 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
RWF-01	Unchecked Value Of ERC20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	● Resolved
SWC-01	**Potential Incorrect Amount Of Tokens Deposited During `compound()`**	**Centralization / Privilege, Logical Issue**	**Major**	● **Acknowledged**
SWC-02	Incompatibility With Deflationary Tokens	Logical Issue	Medium	● Partially Resolved
SWC-03	Divide Before Multiply	Mathematical Operations	Minor	● Resolved
SWC-04	Potential Loss Of Reward	Logical Issue	Minor	● Acknowledged
SWC-05	Unable To Claim Rewards Once LP Tokens Are Withdrawn	Inconsistency, Logical Issue	Minor	● Resolved
SWF-01	**Centralization Risks In Staking.Sol**	**Centralization / Privilege**	**Major**	● **Acknowledged**
SWF-02	Third Party Dependencies	Volatile Code	Minor	● Acknowledged
WCP-01	Missing Zero Address Validation	Volatile Code	Minor	● Resolved
WCP-02	Usage Of `transfer` / `send` For Sending Ether	Volatile Code	Minor	● Resolved

ID	Title	Category	Severity	Status
RWC-01	Missing Emit Events	Coding Style	Informational	● Resolved
SWC-08	Unused Return Value	Volatile Code	Informational	● Resolved

RWF-01 | UNCHECKED VALUE OF ERC20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/Rewarder.sol (base): 49	● Resolved

Description

The linked `transfer()` invocation does not check the return value of the function call which should yield a `true` result in case of a proper `ERC20` implementation.

```
49            reward.transfer(to, amount);
```

Recommendation

As many tokens do not follow the `ERC20` standard faithfully, they may not return a `bool` variable in this function's execution meaning that simply expecting it can cause incompatibility with these types of tokens. Instead, we advise that OpenZeppelin's `SafeERC20.sol` implementation is utilized for interacting with the `transfer()` and `transferFrom()` functions of `ERC20` tokens. The OZ implementation optionally checks for a return value, rendering it compatible with all `ERC20` token implementations.

Alleviation

`[CertiK]` : The team heeded the recommendation and made the changes outlined above in commit 784509d523f8709ddc02ceef196a96fdd9a27a8e.

SWC-01 | POTENTIAL INCORRECT AMOUNT OF TOKENS DEPOSITED DURING `compound()`

Category	Severity	Location	Status
Centralization / Privilege, Logical Issue	● **Major**	**Staking.sol: 151, 199, 852~859, 862**	● **Acknowledged**

Description

In the functions `add()` and `set()`, the `_owner` can set the path of token addresses used to exchange a user's pending reward tokens for the LP token in function `compound()`. If the last token in the path is not set to the pool's LP token, the `compound()` function will swap a user's pending reward token for the incorrect last token. The amount of incorrect tokens is then used as the `amount` argument for function `_deposit()`. The call to `_deposit()` will incorrectly update the users amount of LP tokens, while the balance of LP tokens in the contract does not increase.

Recommendation

We recommend adding a check to the path of addresses such that the last token address is the same address as the pool's LP token.

Alleviation

`[CertiK]` The team acknowledged the issue and made changes in commit f5ae84929405fbe5fc68031fe50c588b33ded94d. However, the addition of the call to `router.getAmountsOut()` does not perform the necessary check outlined above. `getAmountsOut()` can still successfully execute even if the path of addresses is such that the last token address is not the same address as one used in the pair.

SWC-02 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS

Category	Severity	Location	Status
Logical Issue	● Medium	Staking.sol: 590, 651, 852~859	● Partially Resolved

Description

When transferring or swapping deflationary `ERC20` tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrived to the contract. However, a failure to discount such fees may allow the same user to withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

Reference: Gocerberus Exploit

Recommendation

We recommend regulating the set of tokens supported and adding necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

`[CertiK]` The team partially resolved the issue and made changes in commit f5ae84929405fbe5fc68031fe50c588b33ded94d. The new check in function `deposit()` ensure that a pool's LP token cannot be a deflationary token. However, the addition of the check `getAmountsOut()` and `swapExactTokensForTokens()` values being equal, is not a sufficient check for deflationary tokens in the swapping path.

Function `getAmountsOut()` does not account for deflationary tokens because it only checks the reserves of tokens in each pool. Function `swapExactTokensForTokens()` also doesn't account for the transfer fees associated with deflationary tokens, unlike the router function `swapExactTokensForTokensSupportingFeeOnTransferTokens()` .

SWC-03 | DIVIDE BEFORE MULTIPLY

Category	Severity	Location	Status
Mathematical Operations	● Minor	Staking.sol: 844~845, 855	● Resolved

Description

Performing integer division before multiplication truncates the low bits, losing the precision of calculation.

Recommendation

We recommend applying multiplication before division to avoid loss of precision.

Alleviation

`[CertiK]` : The team heeded the recommendation and made the changes outlined above in commit 2e4f3c9ed9c8836209739a6ab9378e3b1f9ba8fe.

SWC-04 | POTENTIAL LOSS OF REWARD

Category	Severity	Location	Status
Logical Issue	● Minor	Staking.sol: 625, 684	● Acknowledged

Description

If a user wishes to claim their previous pending rewards while calling functions `_deposit()` and `withdraw()`, they will not receive any new pending rewards. The amount of reward tokens being transferred lacks the addition of the new pending rewards calculated by the lines 617 to 619.

```
617             uint256 accumlatedReward = ((user.amount + mpInfo.staked) *
618                 pool.accRewardPerShare) / ACC_REWARD_PRECISION;
619             uint256 pending = accumlatedReward - user.rewardDebt;    // Calculate new amount of pending rewards
620
621             if (claimReward) {             // User wishes to claim pending rewards
622                 rewarder[pid].onReward(    // rewarder[pid] contract will transfer `user.pendingReward` amount of reward tokens to user
623                     pool.rewardToken,
624                     to,
625                     user.pendingReward,    // Does not account for the new pending tokens calculated by local var. `pending`
626                     pool.isRewardNative
627                 );
```

Recommendation

We recommend providing more documentation or explaining the intended implementation of the contract such that the current logic can be verified to be correct.

Alleviation

`[CertiK]` The team acknowledged the issue and made changes in commit f5ae84929405fbe5fc68031fe50c588b33ded94d. However, calling the functions `set()` and `add()` with `claimReward` set to true still does not claim/withdraw the new amount of pending rewards for a user.

SWC-05 | UNABLE TO CLAIM REWARDS ONCE LP TOKENS ARE WITHDRAWN

Category	Severity	Location	Status
Inconsistency, Logical Issue	● Minor	Staking.sol: 663, 740~755	● Resolved

Description

A user can call function `withdraw()` to withdraw their deposited LP tokens from a staking pool and choose to claim any pending rewards tokens.

If the user withdraws all their LP tokens from a staking pool and chooses not to claim the pending reward tokens, the reward tokens will become irretrievable. This is because the function `claim()` requires the user to have a non-zero amount of staked LP tokens in the pool in order to claim pending rewards.

Recommendation

Although a user can bypass this issue by re-depositing LP tokens, claiming their pending rewards, and withdrawing the LP tokens, we recommend reworking this logic to ensure a user's rewards can be claimed without depositing more LP tokens.

Alleviation

`[CertiK]` : The team heeded the recommendation and made the changes outlined above in commit 4f4ad39b598e12acaa4ffd004366755fc7ebbdbc.

SWF-01 | CENTRALIZATION RISKS IN STAKING.SOL

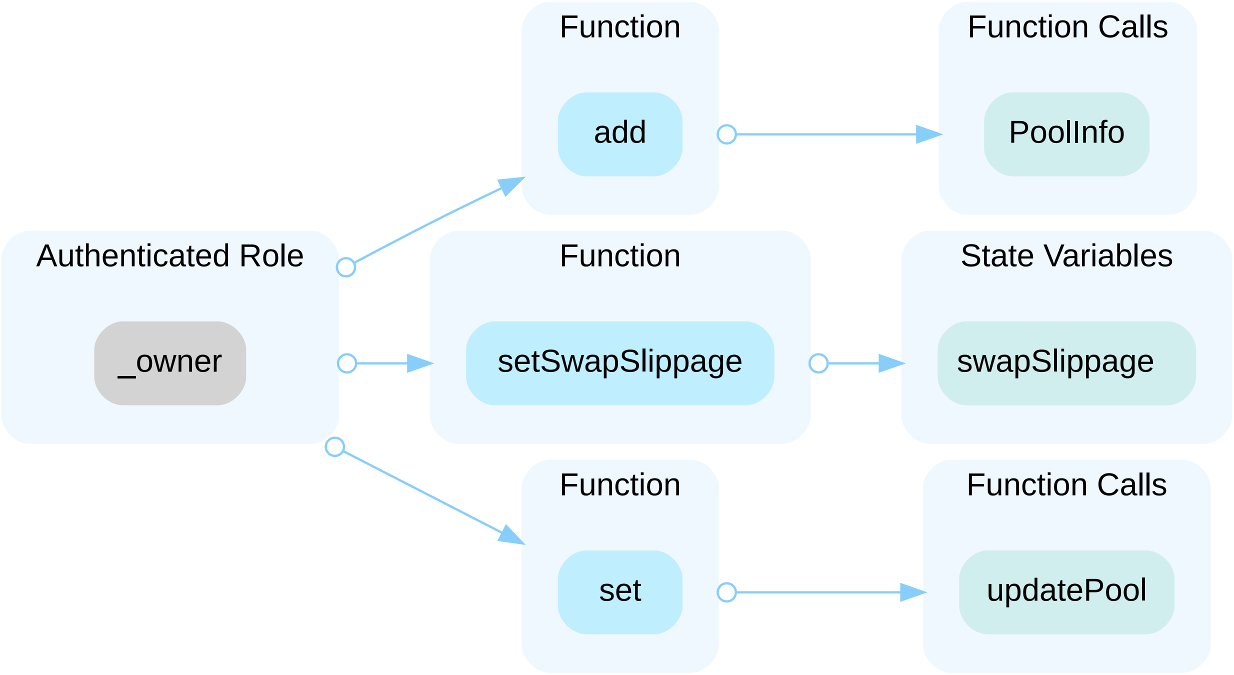
Category	Severity	Location	Status
Centralization / Privilege	● **Major**	**contracts/Staking.sol (base): 128, 182, 211**	● **Acknowledged**

Description

In the contract `Staking` 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:

- Add a new staking pool with function `add()` . For each new pool, the `_owner` can specify:
  - The LP token to be staked in the pool.
  - The reward tokens users will receive for staking the specified LP tokens.
  - The external `rewarder` contract which is in charge of distributing the pool rewards.
  - The path of addresses used to swap a user's pending reward tokens for the last token address in the path and re-deposit the tokens into the same pool.
  - If the pool is initially locked. A locked pool does not allow users to claim or compound their pending rewards, or withdraw their LP tokens from the pool. Furthermore, it does not allow the `_owner` to modify a pool's settings with function `set()` .
  - If the multiplier point functionality is enabled, increasing the pool's reward distribution rate.
  - The privildeged address `breaker` , which can lock/unlock this specific pool. A locked pool does not allow users to claim or compound their pending rewards, or withdraw their LP tokens from the pool. Furthermore, it does not allow the `_owner` to modify a pool's settings with function `set()` .
  - The privildeged address `breakerSetter` , which can set both the `breaker` and `breakerSetter` address to any address for this specific pool.
- Modify the values for any existing pool with function `set()` . For any pool, the `_owner` can change:
  - The external `rewarder` contract to any address. The new `rewarder` contract may not contain enough rewards to distribute to pending users.
  - The path of addresses used to swap a user's pending reward token for the last token address in the path and re-deposit the tokens into the same pool.

- Set the swapping slippage amount to any percent between 0% and 100%, with function `setSwapSlippage()`. The slippage amount is used in the `compound()` function to swap a user's pending reward tokens for another token, and re-deposit the tokens into the pool.



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*.

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

Alleviation

`[Wemix]` : "We will adapt the multi-signature scheme to the contract owner's signature algorithm as a short-term solution and apply DAO as a long-term solution."

SWF-02 | THIRD PARTY DEPENDENCIES

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/Staking.sol (base): 28, 38, 588, 590, 715, 718, 838~843, 847~850, 852~859	● Acknowledged

Description

The contract is serving as the underlying entity to interact with one or more third party protocols. The scope of the audit treats third party entities as black boxes and assume their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

```
28        IERC20[] public lpToken;
```

- The contract `Staking` interacts with third party contracts with `IWWEMIX` interface via `lpToken` .

```
38        IWeswapRouter public router;
```

- The contract `Staking` interacts with third party contracts with `IWeswapRouter` interface via `router` .

Recommendation

We understand that the business logic requires interaction with the third parties. We encourage the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

Alleviation

`[Wemix]` : "Issue acknowledged. We'll constantly monitor the status of third parties to mitigate dependencies."

WCP-01 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	Rewarder.sol: 29, 30; Staking.sol: 109, 129, 130, 131, 184	● Resolved

Description

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

Recommendation

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

Alleviation

`[CertiK]` : The team heeded the recommendation and made the changes outlined above in commit 4e9e4f56d417d9768c26f3e8cf8f8b9775e12aec.

WCP-02 | USAGE OF `transfer` / `send` FOR SENDING ETHER

Category	Severity	Location	Status
Volatile Code	● Minor	Rewarder.sol: 43; Staking.sol: 716	● Resolved

Description

Using Solidity's `transfer()` and `send()` functions for transferring Ether is not recommended, since some contracts may not be able to receive the funds. The cited functions forward only a fixed amount of gas (2300 specifically) and the receiving contracts may run out of gas before finishing the transfer. Also, EVM instructions' gas costs may increase in the future. Thus, some contracts that can receive now may stop working in the future due to the gas limitation.

```
43              to.transfer(amount);
```

- Function `onReward()` uses `transfer()` in contract `Rewarder.sol`

```
716             to.transfer(amount);
```

- Function `withdraw()` uses `transfer()` in contract `Staking.sol`

Recommendation

We recommend using the `Address.sendValue()` function from OpenZeppelin.

Alleviation

`[CertiK]` : The team heeded the recommendation and made the changes outlined above in commit 4f4ad39b598e12acaa4ffd004366755fc7ebbdbc.

RWC-01 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	● Informational	Rewarder.sol: 55	● Resolved

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

We recommend emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

`[CertiK]` : The team heeded the recommendation and made the changes outlined above in commit db7d5a5e27624e53b0a09009fb76e3bd22e8cfcb.

SWC-08 | UNUSED RETURN VALUE

Category	Severity	Location	Status
Volatile Code	● Informational	Staking.sol: 847	● Resolved

Description

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

Recommendation

We recommend checking or using the return values of all external function calls.

Alleviation

`[CertiK]` : The team heeded the recommendation and made the changes outlined above in commit 045df1f1fff5f3dc4f20e1fe1e1a6982caf34ee1.

OPTIMIZATIONS | WEMIX STAKING (GRAND STAKING, DIOS STAKING)

ID	Title	Category	Severity	Status
SWC-06	Comparison To A Boolean Constant	Gas Optimization	Optimization	● Resolved
SWC-07	Unused/Redundant Code Components	Gas Optimization, Logical Issue	Optimization	● Resolved
SWF-03	Improper Usage Of `public` And `external` Type	Gas Optimization	Optimization	● Partially Resolved
SWF-04	State Variables That Could Be Declared As `constant`	Gas Optimization	Optimization	● Resolved

SWC-06 | COMPARISON TO A BOOLEAN CONSTANT

Category	Severity	Location	Status
Gas Optimization	● Optimization	Staking.sol: 192, 232, 250, 263, 274, 612, 673, 738, 773, 808	● Resolved

Description

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

Recommendation

We recommend removing the comparison to the boolean constant.

For example, the following line:

```
192         require(pool.lock == false, "STAKING: EMERGENCY!");
```

can be changed to:

```
192         require(!pool.lock, "STAKING: EMERGENCY!");
```

Alleviation

`[CertiK]`: The team heeded the recommendation and made the changes outlined above in commit b8936be8767d22fd918ba9960375d95bad2cd965.

SWC-07 | UNUSED/REDUNDANT CODE COMPONENTS

Category	Severity	Location	Status
Gas Optimization, Logical Issue	● Optimization	Staking.sol: 56, 373~378, 557~562, 768	● Resolved

Description

The linked code component are never used in the contract or are redundant.

- Variable `MP_PRECISION` is declared but it is never used within the contract.
- The arithmetic logic used to calculate `mpAmount` and `increasedMP` multiplies the calculated value by `multiplierPointBasis` (`1000`), but immediately divides by `BASIS_POINTS_DIVISOR` (`1000`). Since both `multiplierPointBasis` and `BASIS_POINTS_DIVISOR` are set to the value `1000` , this calculation does not affect the overall result in `mpAmount` and `increasedMP` .
- Function `claimWithSwap()` preforms the exact same functionality as function `claim()` .

Recommendation

We recommend removing the unused or unnecessary code components.

- Variable `MP_PRECISION` can be removed from the contract.
- The multiplication of value `multiplierPointBasis` and division of value `BASIS_POINTS_DIVISOR` can be removed from the calculations in `mpAmount` and `increasedMP` .
- Function `claimWithSwap()` can be removed from the contract.

Alleviation

`[Wemix]` : "Issue acknowledged. We removed the unused value `MP_PRECISION` . And now we have the setter to `multiplierPointBasis` , therefore it can be a different value. Finally, we implemented `claimWithSwap` ."

`[CertiK]` : The team resolved this finding in commit 922af9ca6e43c579a5095b695e14318421d6ef30.

The variable `MP_PRECISION` has been removed, a setter function for `multiplierPointBasis` has been added, and `claimWithSwap()` has been implemented. Note the new implementation of `claimWithSwap()` has not been evaluated by CertiK.

SWF-03 | IMPROPER USAGE OF `public` AND `external` TYPE

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/Staking.sol (base): 128, 362, 663, 733, 768 , 803	● Partially Resolved

Description

`public` functions that are never called by the contract could be declared as `external` . `external` functions are more efficient than `public` functions.

```
128        function add(...) public
```

```
362        function pendingMP(uint256 pid, address account) public
```

```
663        function withdraw(...) public
```

```
733        function claim(uint256 pid, address to) public
```

```
768        function claimWithSwap(uint256 pid, address to) public
```

```
803        function compound(uint256 pid, address to) public
```

Recommendation

We recommend using the `external` attribute for `public` functions that are never called within the contract.

Alleviation

`[CertiK]` : The team heeded the recommendation and made the changes outlined above in commit f009b4584c3f017c35eb003b16acb1fff5dc9f5b.

Note, however, a function `setMultiplierPointBasis()` has since been added which is of `public` visibility and should be set to `external` .

SWF-04 | STATE VARIABLES THAT COULD BE DECLARED AS `constant`

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/Staking.sol (base): 54	● Resolved

Description

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

Recommendation

We recommend declaring these variables as `constant`.

Alleviation

`[CertiK]`: The team resolved the finding by implementing a setter function for the variable in commit 40c409040f8568a3fff42cd8da35444dba188f2a. Note that this variable may be set to any `uint256` value by the owner.

APPENDIX | WEMIX STAKING (GRAND STAKING, DIOS STAKING)

Finding Categories

Categories	Description
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.
Mathematical Operations	Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
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.

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 | CERTIK

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.

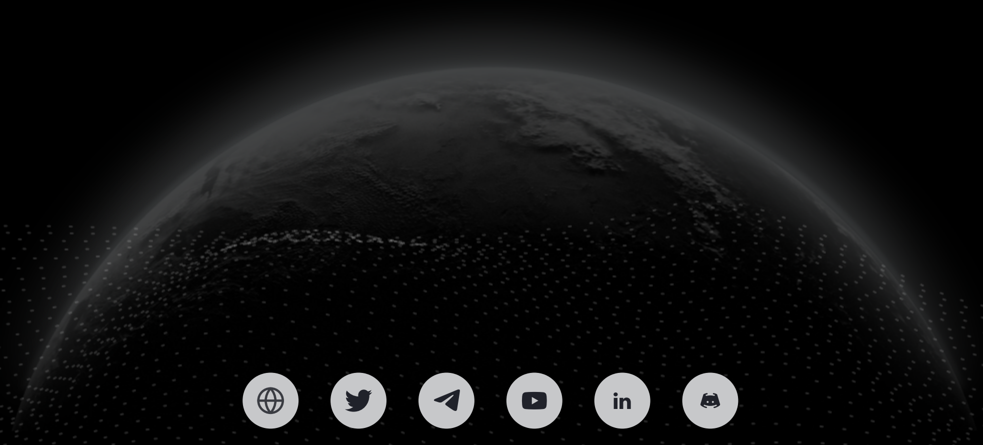
NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND 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.

CertiK | **Securing** the **Web3** World

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.



WEMIX Staking (GRAND Staking, DIOS Staking) Security Assessment | CertiK Verified on Oct 13th, 2022 |