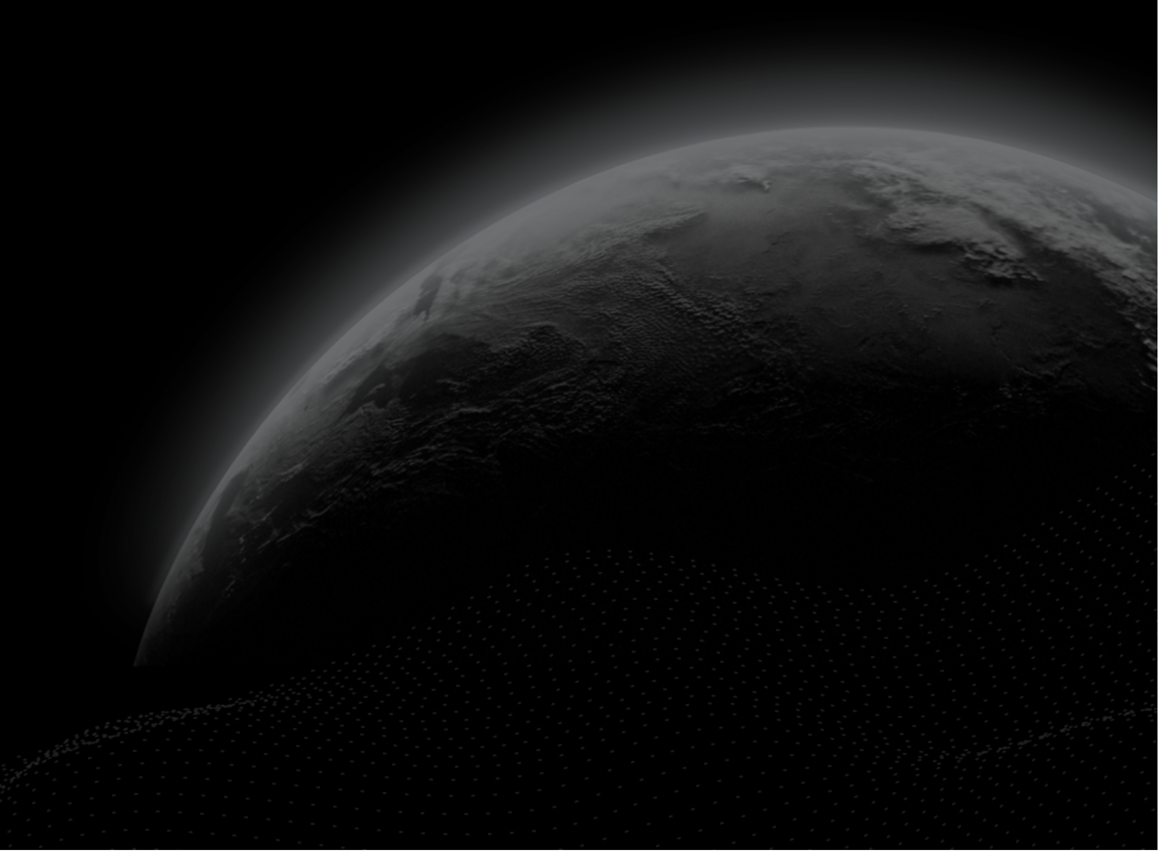
CERTIK

Security Assessment

Marblex

CertiK Assessed on Aug 10th, 2023



CertiK Assessed on Aug 10th, 2023

Marblex

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

Executive Summary

TYPES
ERC-20

ECOSYSTEM
Aurora (AURORA) | Binance Smart Chain (BSC) | Klaytn (KLAY)

METHODS
Manual Review, Static Analysis

LANGUAGE
Solidity

TIMELINE
Delivered on 08/10/2023

KEY COMPONENTS
N/A

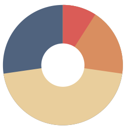
CODEBASE
https://github.com/MarblexAudit/MBXToken-ERC20
View All in Codebase Page

COMMITS
base: 7f5f5149e143f97b5ef728d432873255534a70005
update 1: d4302c2d89369e99b154b4c3cee7f7cb727878f0
update 2: 4fc55ab894fbab42c5ab7926abb26036169fa758
View All in Codebase Page

Highlighted Centralization Risks

- ⓘ Transfers can be paused
- ⓘ Privileged role can mint tokens
- ⓘ Has blacklist/whitelist

Vulnerability Summary



11	7	0	1	3	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

🟥	1	Critical	1 Resolved	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.
🟨	0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨	5	Minor	3 Resolved, 1 Partially Resolved, 1 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.

■	3	Informational	3 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 | MARBLEX

Disclaimer

CODEBASE | MARBLEX

Repository

https://github.com/MarblexAudit/MBXToken-ERC20

Commit

base: 7f5f5149e143f97b5ef728d43287325534a70005

update 1: d4302c2d89369e99b154b4c3cee7f7cb727878f0

update 2: 4fc55ab894fbab42c5ab7926abb26036169fa758

update 3: 9b9a373daff6c3fb066050a3a275458905486f40

update4: 1acafc443daac7fbdaeed3337e5025d1a1717661

AUDIT SCOPE | MARBLEX

4 files audited ● 2 files with Acknowledged findings ● 1 file with Partially Resolved findings ● 1 file without findings

ID	File	SHA256 Checksum
● MBX	MBXToken.sol	fdb9f05ddf22acdf3ddf5da4d6bd089147409b3ce400142232031ca33c186ee8
● MSW	MultiSigWallet.sol	19da476a7c5aed0fd0c34722d6f0b8c8159c88b839a4c52c5e516cdef4eb81de
● TFM	TokenForwarder.sol	c67c77292adf5e8dd33b8d8894384eefddec5d04818dfc41c6d5ccbde8e8be9e
● ERC	ERC2771.sol	1c70a7577c53e9c227747eb7dbb7597726bd827d12f8b75c2471c92cacfef6ca

APPROACH & METHODS | MARBLEX

This report has been prepared for Marblex to discover issues and vulnerabilities in the source code of the Marblex 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.

DEPENDENCIES | MARBLEX

Assumptions

Within the scope of the audit, assumptions are made about the intended behavior of the protocol in order to inspect consequences based on those behaviors. Assumptions made within the scope of this audit include:

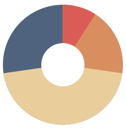
`MBXToken.sol`

- The `trustedForwarder` to be used with the `MBXToken` contract is the in-scope contract `TokenForwarder` of file `TokenForwarder.sol`.
- The `MultiSigWallet` is to be used with the privileged roles of the `MBXToken` contract.

Recommendations

We recommend constantly monitoring the third parties involved to mitigate any side effects that may occur when unexpected changes are introduced. Additionally, we recommend all out-of-scope dependencies are carefully vetted to ensure they function as intended. Last, we recommend all assumptions about the behavior of the project are thoroughly reviewed and, if the assumptions do not match the intention of the protocol, documenting the intended behavior for review.

FINDINGS | MARBLEX



11	1	2	0	5	3
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Marblex. Through this audit, we have uncovered 11 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
MBM-01	Lack Of Access Control	Access Control, Logical Issue	Critical	Resolved
MBX-04	**Centralization Risks In MBXToken.Sol**	**Centralization**	**Major**	**Acknowledged**
MBX-05	**Initial Token Distribution**	**Centralization**	**Major**	**Acknowledged**
MBT-02	Potential Reentrancy Attack (Out-Of-Order Events)	Concurrency	Minor	Partially Resolved
MBX-10	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Acknowledged
TFM-01	Potential Locked Blockchain Native Tokens	Logical Issue	Minor	Resolved
TFM-02	Destination Of `execute()` Can Be Any Address	Access Control	Minor	Resolved
TFM-03	Missing Zero Address Validation	Volatile Code	Minor	Resolved
IMB-01	Unused Event	Coding Issue	Informational	Resolved
MBX-06	Unnecessary Use Of `super` Keyword	Coding Style	Informational	Resolved
MBX-11	Consider Added Checks With `notFrozen` Modifier	Coding Style	Informational	Resolved

MBM-01 | LACK OF ACCESS CONTROL

Category	Severity	Location	Status
Access Control, Logical Issue	● Critical	MBXToken.sol (update1): 153~156	● Resolved

Description

The changes made in commit d4302c2d89369e99b154b4c3cee7f7cb727878f0 introduce a lack of access control on a critically privileged function:

```
function grantRole(bytes32 role, address account) public override(AccessControl,
IAccessControl) {
        _beforeSetRole(role, account, true);
        super._grantRole(role, account);
    }
```

This introduction allows anyone to call the function `grantRole()` because the override calls `super._grantRole()` instead of `super.grantRole()`. Since the external `super.grantRole()` is where the access protection is located, this function can now be called by anyone. In turn, anyone can take on the `MINTER_ROLE` and `PAUSER_ROLE`.

Recommendation

We recommend calling `super.grantRole()` instead of `super._grantRole()` to include the proper protection on the function override.

Alleviation

`[CertiK]`: The team made changes resolving the finding in commit 4fc55ab894fbab42c5ab7926abb26036169fa758.

MBX-04 | CENTRALIZATION RISKS IN MBXTOKEN.SOL

Category	Severity	Location	Status
Centralization	● **Major**	**MBXToken.sol (base): 50, 59, 95~96, 103~104, 108, 134, 142, 156~157, 171, 186~187, 204**	● **Acknowledged**

Description

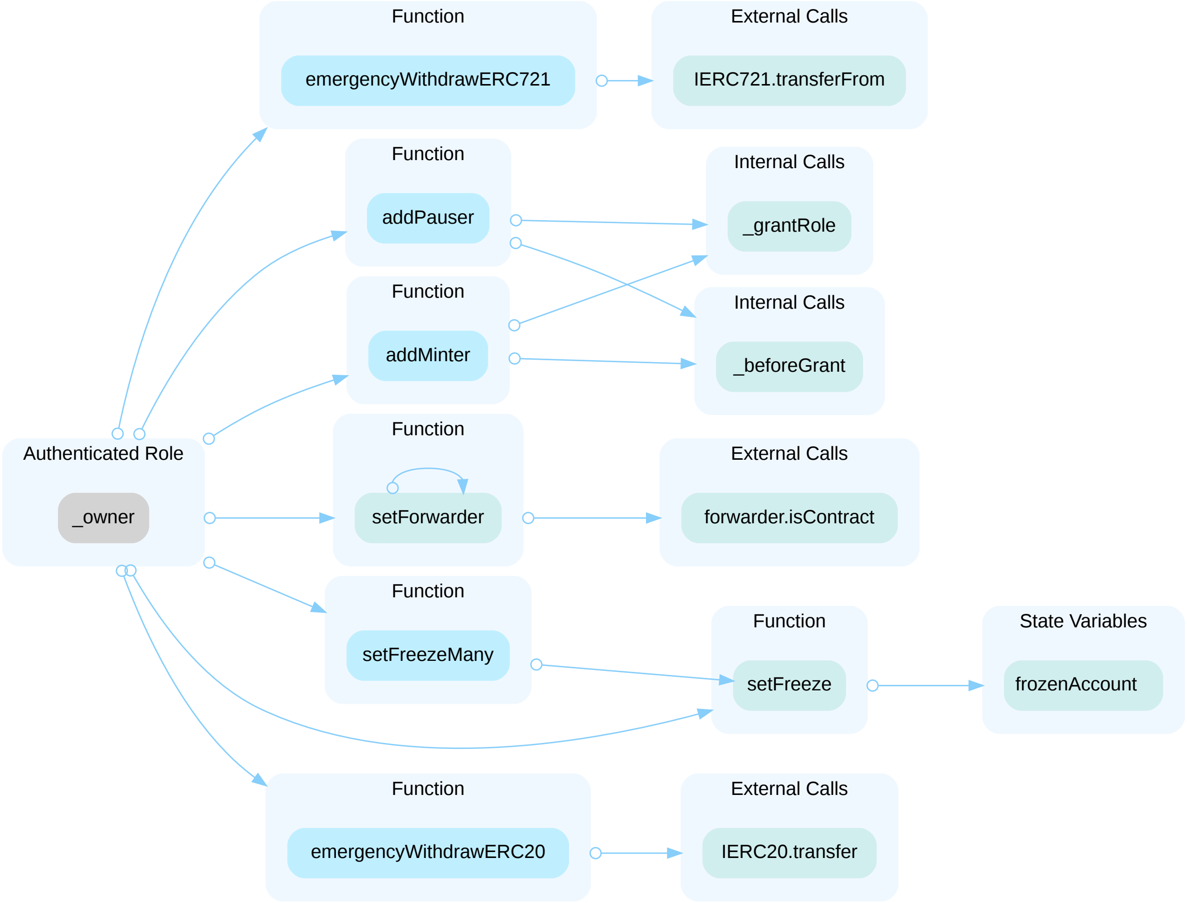
In the contract `MBXToken` the role `_owner` has authority over the functions shown in the diagram below.

Additionally, the `_owner` has authority over the following functions:

- `transferOwnership()`

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and

- set the `trustedForwarder` to one that contains malicious logic in updating the `_msgSender()` within the `MBXToken` contract, possibly allowing for the stealing of funds from users;
- freeze any account to prevent user interaction with their funds;
- add accounts to the `MINTER_ROLE` allowing these accounts to mint any amount of tokens to any address;
- add accounts to the `PAUSER_ROLE` allowing these accounts to pause any functionality in the contract that includes the modifier `whenNotPaused` , including
    - all transfer functions
    - all approval functions
    - all burning functions
- remove accounts from the `MINTER_ROLE` or `PAUSER_ROLE` preventing the intended use of these roles;
- withdraw any ERC20 or ERC721 token sent to the contract;
- transfer the following privileged roles to one account through `acceptOwnership()` , giving all access control to one malicious authority
    - `_owner`
    - `DEFAULT_ADMIN_ROLE`
    - `MINTER_ROLE`
    - `PAUSER_ROLE`



In the contract `MBXToken` the role `DEFAULT_ADMIN_ROLE` has authority over the following functions:

- `grantRole()`
- `revokeRole()`
- `revokeMinter()`
- `revokePauser()`

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and give access to the `MINTER_ROLE` or `PAUSER_ROLE` , allowing any amount of tokens to be minted to any account or pausing the contract to prevent interaction. Additionally, the attacker may use the authority to remove a legitimate account's ability to pause the contract during malicious takeover.

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 privileged roles or removing the function can be considered *fully resolved*.

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

Alleviation

`[CertiK]` : The team states they plan to deploy the token on BSC as a bridged token for their MBX Token currently deployed on Klaytn at the following address.

Klaytn MBX Token: 0xd068c52d81f4409b9502da926ace3301cc41f623

They further state that only their bridge contract will be given the `MINTER_ROLE` , and that the initial minted amount on deploy will be 0.

The team made updates mitigating some of the centralization related risk, by removing functions `emergencyWithdrawERC721()` and `emergencyWithdrawERC20()` , in commit

1acafc443daac7fbdaeed3337e5025d1a1717661.

MBX-05 | INITIAL TOKEN DISTRIBUTION

Category	Severity	Location	Status
Centralization	● **Major**	**MBXToken.sol (base): 43~44**	● **Acknowledged**

Description

All of the `MBXToken` are sent to the contract deployer when deploying the contract, where the deployer specifies the amount to be minted. This could be a centralization risk as the deployer can distribute tokens without obtaining the consensus of the community. Any compromise to the deployer account that holds undistributed tokens may allow the attacker to steal and sell tokens on the market, resulting in severe damage to the project.

Recommendation

We recommend transparency regarding the initial token distribution process. The token distribution plan should be published in a public location that the community can access. The team should also make an effort to restrict the access of the private key. A multi-signature (e.g. ⅔, ⅗) wallet can be used to prevent a single point of failure due to the private key compromise. Additionally, the team can lock up a portion of tokens, release them with a vesting schedule for long-term success, and deanonymize project teams with a third-party KYC provider to create greater accountability.

Alleviation

`[CertiK]` : The team states they plan to deploy the contract with a mint amount of 0.

MBT-02 | POTENTIAL REENTRANCY ATTACK (OUT-OF-ORDER EVENTS)

Category	Severity	Location	Status
Concurrency	● Minor	MultiSigWallet.sol (base): 101, 104; TokenForwarder.sol (base): 39~41, 61	● Partially Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

This finding is considered minor because the reentrancy only causes out-of-order events.

External call(s)

```
 101          (bool success, ) = transaction.to.call{value: transaction.value}(
transaction.data);
```

Events emitted after the call(s)

```
 104          emit ExecuteTransaction(msg.sender, _txIndex);
```

External call(s)

```
39          (bool success, bytes memory returndata) = req.to.call{gas: req.gas,
 value: req.value}(
40              abi.encodePacked(req.data, req.from)
41          );
```

Events emitted after the call(s)

```
61          emit MetaTransactionExecuted(req.from, req.to, req.data);
```

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

`[CertiK]` : The team made changes partially resolving the finding in commit d4302c2d89369e99b154b4c3cee7f7cb727878f0.

Check-effect-interaction pattern is still violated in the function cited within the `MultiSigWallet` contract. It is noted that the function cited is privileged, making it unlikely reentrancy will be accomplished.

The team states they acknowledge the remaining issue and plan to make changes in the future which will not be included presently.

MBX-10 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	● Minor	MBXToken.sol (base): 135	● Acknowledged

Description

The return values of the `transfer()` and `transferFrom()` calls in the smart contract are not checked. Some ERC-20 tokens' transfer functions return no values, while others return a bool value, they should be handled with care. If a function returns `false` instead of reverting upon failure, an unchecked failed transfer could be mistakenly considered successful in the contract.

```
135         IERC20(token).transfer(to, amount);
```

Recommendation

We recommend using the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if false is returned, making it compatible with all ERC-20 token implementations.

Alleviation

`[CertiK]` : The team acknowledges the finding and opts not to change the current version.

TFM-01 | POTENTIAL LOCKED BLOCKCHAIN NATIVE TOKENS

Category	Severity	Location	Status
Logical Issue	● Minor	TokenForwarder.sol (base): 33~34	● Resolved

Description

Function `execute()` of contract `TokenForwarder` is payable, but there is no check that the included `msg.value` matches the input `req.value`. As a result, one of the following scenarios could occur:

- `req.value` is 0, but a positive `msg.value` is included, resulting in native tokens left in the contract.
- If the contract does retain any native tokens, either through the scenario above, or by any other means, then a user can provide a valid signed message that they create, with `req.value` specified as the amount left in the contract. In this case, the caller of `execute()` does not have to provide a `msg.value`, and whatever is left in the contract will be sent wherever the caller specified with their signed message.

Recommendation

We recommend requiring that the `msg.value` matches the `req.value`.

Alleviation

`[CertiK]`: The team made changes resolving the finding in commits

- d4302c2d89369e99b154b4c3cee7f7cb727878f0
- 4fc55ab894fbab42c5ab7926abb26036169fa758
- 9b9a373daff6c3fb066050a3a275458905486f40

TFM-02 | DESTINATION OF `execute()` CAN BE ANY ADDRESS

Category	Severity	Location	Status
Access Control	● Minor	TokenForwarder.sol (base): 39~41	● Resolved

Description

The main use case of `TokenForwarder` appears to be its role as the `trustedForwarder` address used in the `ERC2771` inheritance of the `MBXToken` contract. If there are no other use cases of this contract, consider setting the destination address for the low-level `call` in the function `execute()` upon deployment of the contract, instead of letting the user determine the destination.

With its current set up, users can sign any message for any `req.to` destination, and the contract will execute the call to that destination.

Recommendation

We recommend considering the restriction of the potential interactions that can take place with the `TokenForwarder` contract, if its only intended use is with the `MBXToken` contract.

Alleviation

`[CertiK]` : The team made changes resolving the finding in commit d4302c2d89369e99b154b4c3cee7f7cb727878f0.

TFM-03 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	TokenForwarder.sol (base): 39	● Resolved

Description

The `to` address is not validated before assignment or external calls, potentially allowing the use of the zero address and leading to unexpected behavior or vulnerabilities. For example, transferring tokens to a zero address can result in a permanent loss of those tokens.

Recommendation

We recommend adding a check that the passed-in address in `execute()` is not `address(0)` to prevent unexpected errors.

Alleviation

`[CertiK]` : The team made changes resolving this finding in commit d4302c2d89369e99b154b4c3cee7f7cb727878f0.

IMB-01 | UNUSED EVENT

Category	Severity	Location	Status
Coding Issue	● Informational	interfaces/IMBXToken.sol (base): 7	● Resolved

Description

Some events are never emitted, which can lead to confusion and code maintainability issues.

```
7      event SetStatus(bool enableERC2612, bool enableERC2771);
```

- `SetStatus` is declared in `IMBXToken` but never emitted.

Recommendation

We recommend removing the unused event or emitting it in the intended functions to improve code clarity and maintainability.

Alleviation

`[CertiK]` : The team made changes resolving the finding in commit d4302c2d89369e99b154b4c3cee7f7cb727878f0.

MBX-06 | UNNECESSARY USE OF `super` KEYWORD

Category	Severity	Location	Status
Coding Style	● Informational	MBXToken.sol (base): 53~54, 62~63, 75~76, 88~89, 97~98, 105~106	● Resolved

Description

In the locations cited, the function called is inherited by the contract and can be referenced directly, without the use of the keyword `super`.

Recommendation

We recommend removing the unnecessary use of `super`.

Alleviation

`[CertiK]`: The team made changes resolving the finding in commit d4302c2d89369e99b154b4c3cee7f7cb727878f0.

MBX-11 | CONSIDER ADDED CHECKS WITH `notFrozen` MODIFIER

Category	Severity	Location	Status
Coding Style	● Informational	MBXToken.sol (base): 220~221, 271~272, 279~280	● Resolved

Description

The current implementation of the `MBXToken` only ensures that tokens cannot be transferred `from` an account that has been frozen. There are no checks on `msg.sender` (potentially distinct from `_msgSender()` , or the `to` account, all of which may be different addresses.

- If tokens are transferred `to` a frozen account, then tokens that were previously in circulation become temporarily unavailable, while the account they were transferred to is left frozen.
- In functions `permit()` and `_approve()` , the `spender` address is not checked to ensure the address is not frozen. This may allow a frozen account to send the owner's tokens to maliciously.
- The `_msgSender()` executing any function call may be a frozen account. If a user has previously given approval to an account that becomes frozen, then the frozen account can still use the approval to transfer the tokens to any destination address. Additionally, a frozen `msg.sender` account can still use a valid signature in the `permit()` function on behalf of a non-frozen account.
- In cases where the `tokenForwarder` contract is used to relay an address for `_msgSender()` that is distinct from the `msg.sender` interacting, then the `_msgSender()` can be a frozen account and still make calls to functions `transferFrom()` and `permit()` as described above.

Recommendation

If the above is intended behavior of the protocol, no action is needed, and upon confirmation, the finding will be resolved. Otherwise, we recommend considering the addition of the modifier `notFrozen()` for addresses `to` , `msg.sender` , and `_msgSender()` (in the case where `msg.sender` and `_msgSender()` are distinct from one another).

Alleviation

`[CertiK]` : The team notes that the added checks do not fit the needs of the protocol, so the finding is resolved.

OPTIMIZATIONS | MARBLEX

ID	Title	Category	Severity	Status
MBT-01	Functions Equivalent To Compiler-Generated Getters	Gas Optimization, Code Optimization	Optimization	Acknowledged
MBX-02	Unnecessary Requirements	Gas Optimization, Code Optimization	Optimization	Resolved
MBX-07	Redundant References And Use Of Modifier Checks	Gas Optimization, Code Optimization	Optimization	Partially Resolved
MBX-08	Declaration Of Specific Access Control Functions	Gas Optimization, Code Optimization	Optimization	Resolved
MBX-09	Modifier `notFrozen` Can Be Refactored For Gas Optimization During Deployment	Gas Optimization	Optimization	Resolved
MSW-01	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	Resolved
MSW-02	Inefficient Memory Parameter	Inconsistency	Optimization	Resolved

MBT-01 | FUNCTIONS EQUIVALENT TO COMPILER-GENERATED GETTERS

Category	Severity	Location	Status
Gas Optimization, Code Optimization	● Optimization	MBXToken.sol (base): 243~244; MultiSigWallet.sol (base): 126~127	● Acknowledged

Description

`MBXToken.sol`

Function `getNonce()` returns `super.nonces(from)` , where `from` is a user-provided input.

The mapping `nonces` has a compiler-generated getter function which returns the same output.

`MultiSigWallet.sol`

Function `getTransaction()` returns the same information that is returned from directly referencing the compiler-generated getter function for the `transactions` array.

Recommendation

We recommend relying on the compiler-generated getter functions to reference the respective return values, and removing the functions `getNonce()` and `getTransaction()` from their respective contracts.

Alleviation

`[CertiK]` : The team acknowledges the finding and opts not to make changes to their current version.

They further state that the `getNonce()` function is a wrapper for another project's interface.

MBX-02 | UNNECESSARY REQUIREMENTS

Category	Severity	Location	Status
Gas Optimization, Code Optimization	● Optimization	MBXToken.sol (base): 157~158, 297~298	● Resolved

Description

- Function `acceptOwnership()` overridden in `MBXToken` includes a requirement that `pendingOwner()` is not `address(0)`. However, it is not possible for `address(0)` to call this function directly, and the `TokenForwarder` contract that may be used to change the return value of `_msgSender()` cannot send `address(0)` as the source address. This is because the recovered signer of the `ForwardRequest` is checked in the `ECDSA` library to be a nonzero address, and reverts if this is the case. Consequently, the check that the `pendingOwner()` is not `address(0)` is unnecessary and can be removed.

- Internal function `_beforeGrant()` requires that the input `role` is not the `DEFAULT_ADMIN_ROLE`, however, this internal function is only called in functions `addMinter()` and `addPauser()` where the `role` is either `MINTER_ROLE` or `PAUSER_ROLE` respectively. Consequently, the check that the `role` is not the `DEFAULT_ADMIN_ROLE` is unnecessary and can be removed.

Recommendation

We recommend removing the unneeded requirements.

If the recommendation of finding MBX-08 is followed regarding the use of `_beforeGrant()` in functions `addMinter()` and `addPauser()`, then the check to `DEFAULT_ADMIN_ROLE` is no longer unnecessary and should remain in the function as a valid check.

Alleviation

`[CertiK]`: The team made changes resolving the finding in commit d4302c2d89369e99b154b4c3cee7f7cb727878f0.

MBX-07 | REDUNDANT REFERENCES AND USE OF MODIFIER CHECKS

Category	Severity	Location	Status
Gas Optimization, Code Optimization	● Optimization	MBXToken.sol (base): 220~221, 259~260, 271~272, 272~273, 279~280, 280~281	● Partially Resolved

Description

Hooks `_beforeTokenApprove()` and `_beforeTokenTransfer()` are both used to add the same checks that the `from` address is not frozen (modifier `notFrozen`), and that the contract is not paused (modifier `whenNotPaused`). There are some functions in which both modifiers are called more than once on the same input.

- Inherited function `transferFrom()` uses both hooks because `_spendAllowance()` calls function `_approve()` which is overridden to include `_beforeTokenApprove()`, and because internal `_transfer()` is called which includes `_beforeTokenTransfer()`;
- Function `permit()` is overridden to include modifiers `notFrozen` and `whenNotPaused`, and its inherited logic calls internal function `_approve()`
- Inherited function `burnFrom()` uses both hooks because `_spendAllowance()` calls function `_approve()` which is overridden to include `_beforeTokenApprove()`, and because internal `_burn()` includes `_beforeTokenTransfer()`;

Internal function `_beforeTokenApprove()` also includes a reference to `super._beforeTokenTransfer()` in the body of the function. This references the inherited logic of the `_beforeTokenTransfer()` function, which also includes a check that the contract is not `paused`. The reference to `super._beforeTokenTransfer()` is unnecessary in the body of the `_beforeTokenApprove()` function.

In the override of `_beforeTokenTransfer()` within the `MBXToken` contract, there is also a reference to `super._beforeTokenTransfer()`. Since this logic includes a check that the contract is not paused, it is not necessary to include the modifier `whenNotPaused` in the override of the `_beforeTokenTransfer()` function.

Recommendation

We recommend reworking the logic so that each check is only made once. One solution could be to remove the `whenNotPaused` modifier from the hook `_beforeTokenTransfer()`, and to remove the `_beforeTokenApprove()` hook and replace with modifiers in each of the following external functions:

- `approve()`
- `increaseAllowance()`

- `decreaseAllowance()`

In doing so, all approval functionality will still include the same checks, and the `permit()` , `transferFrom()` , and `burnFrom()` functions will now only include the check once.

Alleviation

`[CertiK]` : The team made changes which partially resolve the finding in commit d4302c2d89369e99b154b4c3cee7f7cb727878f0.

Namely, the function `_beforeTokenTransfer()` was streamlined to only include a check to `whenNotPaused` once and `_beforeTokenApprove()` (renamed `_beforeTokenTransaction()`) now references the override of `_beforeTokenTransfer()` .

However, the functions `transferFrom()` , `permit()` , and `burnFrom()` still include the checks multiple times because of the reasons cited in the description of the finding.

MBX-08 | DECLARATION OF SPECIFIC ACCESS CONTROL FUNCTIONS

Category	Severity	Location	Status
Gas Optimization, Code Optimization	● Optimization	MBXToken.sol (base): 50~51, 59~60, 73~74, 86~87, 95~96, 103~104, 230~231, 235~236	● Resolved

Description

The contract `MBXToken` has a code size that exceeds the limit of 24576 bytes. It is noted that there are several functions added which call existing inherited functions from `AccessControlEnumerable` with hardcoded input:

- `addMinter()`
- `addPauser()`
- `renounceMinter()`
- `renouncePauser()`
- `revokeMinter()`
- `revokePauser()`
- `isMinter()`
- `isPauser()`

Functions `renounceMinter()` , `renouncePauser()` , `revokeMinter()` , `revokePauser()` , `isMinter()` , and `isPauser()` appear unneeded. The `MINTER_ROLE` and `PAUSER_ROLE` values are public and include compiler-generated getter functions. These getter functions can be used to return the bytes32 value representing each role, and then these roles can be used as input, along with the desired account address in functions `renounceRole()` , `revokeRole()` , and `hasRole()` respectively.

The `bytes32` value of `MINTER_ROLE` is 0x9f2df0fed2c77648de5860a4cc508cd0818c85b8b8a1ab4ceeef8d981c8956a6.

The `bytes32` value of `PAUSER_ROLE` is 0x65d7a28e3265b37a6474929f336521b332c1681b933f6cb9f3376673440d862a

Functions `addMinter()` and `addPauser()` include other checks through call to internal `_beforeGrant()` before calling `_grantRole()` . However, it is noted that `DEFAULT_ADMIN_ROLE` can still directly call function `grantRole()` to bypass these checks. Since the `owner` of the contract is necessarily also a `DEFAULT_ADMIN_ROLE` based on the logic of the contract, this makes the addition of the checks in `addMinter()` and `addPauser()` functions ineffectual.

Recommendation

We recommend reducing the size and complexity of the codebase by removing the unneeded functions. Consider removing functions `renounceMinter()` , `renouncePauser()` , `revokeMinter()` , `revokePauser()` , `isMinter()` , and `isPauser()` , and relying on the inherited functions instead.

Consider removing functions `addMinter()` and `addPauser()` replacing with an override of function `grantRole()` which adds the checks in `_beforeGrant()` and then calls `super.grantRole()` . This will ensure that the `DEFAULT_ADMIN_ROLE` and the `_owner` adhere to the `_beforeGrant()` checks while reducing the code size.

If there is a reason for including the functions that pertains to the use of the `trustedForwarder` or the `MultiSigWallet` , please provide documentation on the necessity of the functions listed above.

Alleviation

`[CertiK]` : The team made changes which resolve the finding in commit d4302c2d89369e99b154b4c3cee7f7cb727878f0.

MBX-09 | MODIFIER `notFrozen` CAN BE REFACTORED FOR GAS OPTIMIZATION DURING DEPLOYMENT

Category	Severity	Location	Status
Gas Optimization	● Optimization	MBXToken.sol (base): 28~29	● Resolved

Description

The modifier `notFrozen()` can be reconstructed to save gas during deployment by calling an internal view function instead of directly calling a `require` statement. See an example of this implementation in the following OpenZeppelin contract: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol#L46.

Code explicitly written in modifiers is copied in all other function instances in which the modifier is used within the contract. In turn, the overall size of the contract is increased. This can be prevented by instead using an internal view function for the required check, as is demonstrated in the link above.

Note, however, that function calls in which this is used may cost a slight extra amount in gas each time if this revision is made.

Recommendation

We recommend considering the refactoring of the modifier `notFrozen()` to call an internal view function with the `require` logic incorporated to save gas during deployment of the logic contract.

Alleviation

`[CertiK]` : The team made changes resolving the finding in commit d4302c2d89369e99b154b4c3cee7f7cb727878f0.

MSW-01 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	● Optimization	MultiSigWallet.sol (base): 15	● Resolved

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

Alleviation

`[CertiK]`: The team made changes resolving the finding in commit d4302c2d89369e99b154b4c3cee7f7cb727878f0.

MSW-02 | INEFFICIENT MEMORY PARAMETER

Category	Severity	Location	Status
Inconsistency	● Optimization	MultiSigWallet.sol (base): 75	● Resolved

Description

One or more parameters with `memory` data location are never modified in their functions and those functions are never called internally within the contract. Thus, their data location can be changed to `calldata` to avoid the gas consumption copying from calldata to memory.

```
75      function submitTransaction(address _to, uint256 _value, bytes memory _data)
public onlyOwner {
```

`submitTransaction` has memory location parameters: `_data` .

Recommendation

We recommend changing the parameter's data location to `calldata` to save gas.

- For Solidity versions prior to 0.6.9, since public functions are not allowed to have calldata parameters, the function visibility also needs to be changed to `external` .
- For Solidity versions prior to 0.5.0, since parameter data location is implicit, changing the function visibility to `external` will change the parameter's data location to calldata as well.

Alleviation

`[CertiK]` : The team made changes resolving the finding in commit d4302c2d89369e99b154b4c3cee7f7cb727878f0.

APPENDIX | MARBLEX

Finding Categories

Categories	Description
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.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Concurrency	Concurrency findings are about issues that cause unexpected or unsafe interleaving of code executions.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.

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.