



Security Assessment

EthereumMax Token

Jul 28th, 2021

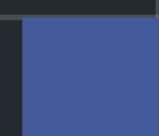
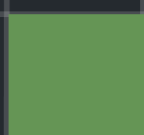


Table of Contents

Summary

Overview

Project Summary

Audit Summary

Vulnerability Summary

Audit Scope

Findings

EMC-01 : Unlocked Compiler Version

MEM-01 : Unlocked Compiler Version

MEM-02 : Division Before Multiplication

REF-01 : Unlocked Compiler Version

REF-02 : Mutability Specifiers Missing

REF-03 : Redundant `array` Look Up

REF-04 : Ambiguous Function Call

Appendix

Disclaimer

About

Summary

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

The auditing process pays special attention to the following considerations:

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

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

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

Overview

Project Summary

Project Name	EthereumMax Token
Description	A typical reflect token fork
Platform	Ethereum
Language	Solidity
Codebase	- EtherScan - GitHub
Commit	25a3aa78e3d847a72b69b90d2315bb09f8e961ea 85d11135a5ea48761b192eefe33feb1aac5bf6c2

Audit Summary

Delivery Date	Jul 28, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	EthereumMax Token

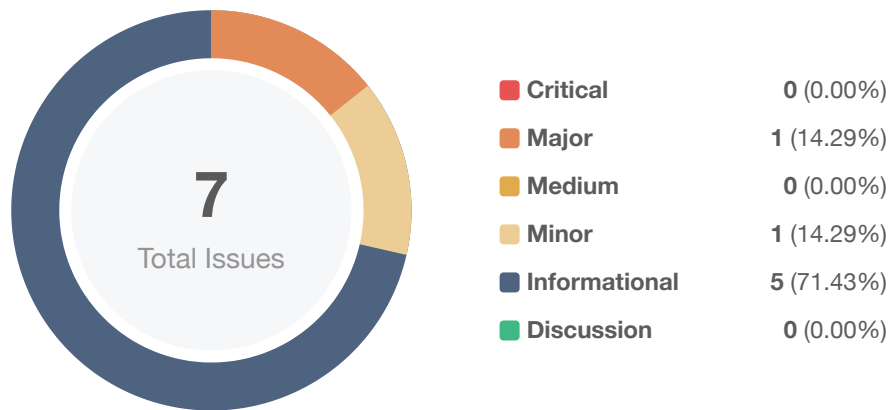
Vulnerability Summary

Vulnerability Level	Total	Pending	Partially Resolved	Resolved	Acknowledged	Declined
● Critical	0	0	0	0	0	0
● Major	1	0	0	1	0	0
● Medium	0	0	0	0	0	0
● Minor	1	0	0	1	0	0
● Informational	5	0	0	2	3	0
● Discussion	0	0	0	0	0	0

Audit Scope

ID	file	SHA256 Checksum
EMC	contracts/EMaxCoin3.sol	8ac3ba878930f55f040e7c324d6bc172f6d65b8d00d071c683101f38529d691c
REF	contracts/REFLECT3.sol	4f08133b2afd936fc6dfac0a691d349cbe4bb43a60e88dd21c333490e729c080

Findings



ID	Title	Category	Severity	Status
EMC-01	Unlocked Compiler Version	Language Specific	Informational	Acknowledged
MEM-01	Unlocked Compiler Version	Language Specific	Informational	Acknowledged
MEM-02	Division Before Multiplication	Mathematical Operations	Minor	Resolved
REF-01	Unlocked Compiler Version	Language Specific	Informational	Acknowledged
REF-02	Mutability Specifiers Missing	Gas Optimization	Informational	Resolved
REF-03	Redundant <code>array</code> Look Up	Gas Optimization	Informational	Resolved
REF-04	Ambiguous Function Call	Logical Issue	Major	Resolved

EMC-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	contracts/EMaxCoin3.sol: 2	ⓘ Acknowledged

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

MEM-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	projects/EthereumMax/eMax.sol (d74ab5548b0bfc30400a11cc4443206094e5d2f8): 7	ⓘ Acknowledged

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

MEM-02 | Division Before Multiplication

Category	Severity	Location	Status
Mathematical Operations	● Minor	projects/EthereumMax/eMax.sol (d74ab5548b0bfc30400a11cc4443206094e5d2f8): 641	☑ Resolved

Description

Integer division might truncate, hence division before multiplication can lead to loss of precision.

Recommendation

We advise to perform multiplication before division to avoid loss of precision.

Alleviation

The development team has acknowledged this exhibit and changed the linked operation by removing the multiplication.

REF-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	contracts/REFLECT3.sol: 3	ⓘ Acknowledged

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

REF-02 | Mutability Specifiers Missing

Category	Severity	Location	Status
Gas Optimization	● Informational	contracts/REFLECT3.sol: 29~31	✓ Resolved

Description

The linked variables are assigned to only once, during the `constructor`'s execution.

Recommendation

We advise that the `immutable` mutability specifier is set at the variable's contract-level declaration to greatly optimize the gas cost of utilizing the variables. Please note that the `immutable` keyword only works in Solidity versions `v0.6.5` and up.

Alleviation

The development team opted to consider our references and changed the mutability of the linked state variables to `immutable`.

REF-03 | Redundant array Look Up

Category	Severity	Location	Status
Gas Optimization	● Informational	contracts/REFLECT3.sol: 191	✓ Resolved

Description

The linked `for` loop conditionals redundantly use the `length` member of the `_excluded` array, as the repeated storage reads increase the gas consumption.

Recommendation

We advise to assign the `array` size to a local variable instead.

Alleviation

The development team opted to consider our references and introduce a local variable for the array size.

REF-04 | Ambiguous Function Call

Category	Severity	Location	Status
Logical Issue	● Major	contracts/REFLECT3.sol: 163	✓ Resolved

Description

The linked function invocation ambiguously uses the contract's address instead of the `sender` parameter, leading to missing event representing the transfer of burn fee from actual sender to contract's address before the call to `_burnTokens()` function. This will result in explorers, such as EtherScan, showing incorrect amounts of tokens as they track balances using events.

Recommendation

We advise to use `sender` instead of `address(this)` in the linked function invocation, as the `sender` argument of the `_burnTokens()` function only serves the event emission.

Alleviation

The development team opted to consider our references and adjusted the function call as proposed.

Appendix

Finding Categories

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.

Language Specific

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

Checksum Calculation Method

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

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

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to 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 without CertiK's prior written consent.

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.

About

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

