PRIVATE

Code Assessment

of the METL Coin Smart Contracts

December 8, 2021

Produced for

METL

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Anna,

Thank you for trusting us to help METL Technologies with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of METL Coin according to Scope to support you in forming an opinion on their security risks.

METL Technologies implements an ERC20 token with extensive access control.

The most critical subjects covered in our audit are functional correctness and access control. One highly important issue was uncovered where frozen users were able to unfreeze themselves. The issue was fixed. Hence, security regarding all the aforementioned subjects is high.

The general subjects covered are upgradability, unit testing and gas efficiency. Security regarding all the aforementioned subjects is high. Moreover, the specification was comprehensive and covered the whole codebase.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	1
• Code Corrected	1
Medium-Severity Findings	2
• Code Corrected	2
Low-Severity Findings	7
• Code Corrected	6
• Acknowledged	1

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on `METL.sol` file based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	2 December 2021	8260216631375ad650eb824aef9d63b7a50cc5a8	Initial Version
2	7 December 2021	dbdc261d7ac41a3584f35f838182e5939f100216	Second Version
3	8 December 2021	04858b547ca36acca1b0940caa8140feffd0f334	Final Version

For the solidity smart contracts, the compiler version `0.8.10` was chosen. The contracts will be deployed on the Avalanche chain.

2.1.1 Excluded from scope

METL Technologies informed us that `poolAddress` will not be used, hence all the functions that modify it are out of scope. Moreover, there are multiple functions that wrap `grantRole`. These functions implement access control even though the `grantRole` already implements the same access control. METL Technologies is aware of these redundancies.

There is extensive use of contracts provided by the Open Zeppelin library. These contracts are assumed to function correctly.

The contracts are deployed on the Avalanche chain which uses EVM. We assume that the contracts are executed in a similar environment to Ethereum.

Finally, we assume that `hardhat/console.sol` library will be removed before deployment.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

At the end of this report section we have added subsections for each of the changes accordingly to the versions.

Furthermore, in the findings section we have added a version icon to each of the findings to increase the readability of the report.

2.2.1 METL

METL Technologies offers an upgradeable ERC20 token that also:

- is burnable: anyone can burn their own or approved tokens
- is pausable: if paused, minting, burning and transferring tokens will revert

- has access control with the following roles: `DEFAULT_ADMIN`, `MINTER`, `BURNER`, `FREEZER`, `FROZEN`, `PAUSER` and `MULTISIG`.

Access control is such that:

- `DEFAULT_ADMIN`: initially the contract deployer. This role can grant and revoke the following roles for any address: `DEFAULT_ADMIN`, `MINTER`, `BURNER`, `FREEZER`, `PAUSER` and `MULTISIG`. An address with this role can change the `poolAddress` and transfer tokens from the `poolAddress` to a designated recipient. The transfer functionality requires the `poolAddress` to have approved the token contract the amount in the first place.
- `MINTER`: can mint tokens to the `poolAddress` and can mint tokens to a recipient that has the `MULTISIG` role.
- `BURNER`: can burn tokens from the `poolAddress`.
- `FREEZER`: can grant and revoke `FROZEN` role to addresses.
- `FROZEN`: no right, used to blacklist addresses. A frozen address cannot send nor receive tokens or even burn their own tokens.
- `PAUSER`: can pause and unpause the contract.
- `MULTISIG`: no rights, used to whitelist addresses that can be minted to.

2.2.2 Trust model:

Addresses with role `DEFAULT_ADMIN_ROLE` are fully trusted and they trust the addresses they grant `MINTER_ROLE`, `BURNER_ROLE`, `FREEZER_ROLE` and `PAUSER_ROLE` in their respective roles. All these addresses are trusted to be non-malicious.

2.2.3 Version 2

For Version 2, `poolAddress` state variable as well as all the function that modified it were removed. Moreover, the `bankBurn` functionality was introduced which allows the `BURNER_ROLE` to burn the tokens of a user.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts associated with the items defined in the engagement letter on whether it is used in accordance with its specifications by the user meeting the criteria predefined in the business specification. We draw attention to the fact that due to inherent limitations in any software development process and software product an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third party infrastructure which adds further inherent risks as we rely on the correct execution of the included third party technology stack itself. Report readers should also take into account the facts that over the life cycle of any software product changes to the product itself or to its environment, in which it is operated, can have an impact leading to operational behaviours other than initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severities. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	1

- Access Control Functionalities Redundancy Acknowledged

5.1 Access Control Functionalities Redundancy

Design Low Version 1 Acknowledged

Every `addROLE_XXX` and `removeROLE_XXX` have redundant checks and events, `grantRole` and `revokeRole` already check that the caller is an admin for `ROLE_XXX` internally and also emit an event while a similar event is emitted from `grantRole`.

Acknowledged:

These functionalities are intentionally duplicated to facilitate front-end development. The events have been removed in Version 2.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	1

- Frozen Users Can Unfreeze Themselves Code Corrected

Medium-Severity Findings	2

- Missing Check for Whitelisting Code Corrected
- grantRole Called on Revoke Code Corrected

Low-Severity Findings	6

- Redundant Check for Admin Code Corrected
- Redundant Revocation Code Corrected
- Code With No Effect Code Corrected
- Compiler Version Not Fixed and Outdated Code Corrected
- Missing Indexes In Events Code Corrected
- Missing Zero Address Check Code Corrected

6.1 Frozen Users Can Unfreeze Themselves

Design High Version 1 Code Corrected

A frozen user can call `renounceRole` from `AccessControlUpgradeable` and unfreeze themselves. Note, that the Open Zeppelin implementation assumes that roles entail privileges and thus, any user can give them up if they want to.

Moreover, the last admin could circumvent the check that enforces at least one admin for the contract by calling `renounceRole`. However, this functionality will not be available in the front end so we do not expect an admin to call `renounceRole` by mistake.

Code corrected:

`renounceRole` has been overriden by `METL` contract, also implementing check to prevent frozen users to unfreeze themselves and last default admins renouncing themselves.

6.2 Missing Check for Whitelisting

Correctness Medium Version 2 Code Corrected

`bankTransfer` allows the admin to send money from the bank to a recipient. The recipient must be whitelisted by having the `MULTISIG_ROLE`. However, the sender is sanitized while the receiver is not. The error message emitted on revert i.e., `Recipient must be whitelisted`, also implies that the receiver should be sanitized.

Code corrected:

The function `bankTransfer` has been removed.

6.3 `grantRole` Called on Revoke

Correctness | Medium | Version 1 | Code Corrected

In `revokeMultisig` the `grantRole` function is called instead of `revokeRole`. Note, however, that the default admin can still revoke the `MULTI_SIG_ROLE` by directly calling `revokeRole`.

Code corrected:

`grantRole` has been replaced with `revokeRole`.

6.4 Redundant Check for Admin

Design | Low | Version 2 | Code Corrected

`removeAdmin` checks whether there are at least two admin users before removing one. However, it makes use of `revokeRole` which is implemented in the same contract and also preforms a similar check.

Code corrected:

The check for number of admins has been removed in `removeAdmin`.

6.5 Redundant Revocation

Design | Low | Version 2 | Code Corrected

Currently, the `METL` contract implements `revokeRole` as shown below.

```
function revokeRole(bytes32 role, address account) public override {
  if (role == DEFAULT_ADMIN_ROLE) {
    require(getRoleMemberCount(role) > 1, "Contract requires one admin");
    super.revokeRole(role, account);
  }
  super.revokeRole(role, account);
}
```

Notice that in case the role to be revoked is the `DEFAULT_ADMIN_ROLE`, `super.revokeRole` will be called twice, with the second call having essentially no effect. Moreover, this redundancy prevents all admins to renounce their own `DEFAULT_ADMIN_ROLE` themselves.

Code corrected:

The call to `super.revokeRole` in the `if` branch has been removed.

6.6 Code With No Effect

Design | Low | Version 1 | Code Corrected

`initialize()` sets `poolAddress` to `address(0)`, but `poolAddress` is already `address(0)` by default on contract creation.

Code corrected:

`poolAddress` as well as its initialization were removed.

6.7 Compiler Version Not Fixed and Outdated

Design | Low | Version 1 | Code Corrected

The solidity compiler is not fixed in the contracts. The version, however, is defined in the `hardhat.config.js` to be `0.8.4`.

In the code the following pragma directive is used:

```
pragma solidity ^0.8.0;
```

Known bugs in version `0.8.4` are:

https://github.com/ethereum/solidity/blob/develop/docs/bugs_by_version.json#L1562

More information about these bugs can be found here:

https://docs.soliditylang.org/en/latest/bugs.html

At the time of writing the most recent Solidity release is version `0.8.10` which contains some bugfixes.

Moreover, in `hardhat.config.js`, the optimizer is not explicitly enabled and the default value for hardhat is `enabled: false`. Enabling the optimizer may help to reduce gas costs. We acknowledge however, for the time being, the gas costs on Avalanche chain, where the contract will be deployed, the difference fee-wise is small.

Code corrected:

The compiler version has been updated to `0.8.10` which is the latest version as the report was written.

The optimizer is still disabled. However, as aforementioned, we do not expect to provide significant improvement.

6.8 Missing Indexes In Events

Design Low Version 1 Code Corrected

The events `RoleChange` and `PoolChange` contain no indexed fields. Indexing fields will help for searching events quicker.

Code corrected:

The events were removed. The current implementation relies on the events emitted by the `AccessControlEnumerableUpgradeable`.

6.9 Missing Zero Address Check

Design Low Version 1 Code Corrected

In `changePoolAddress`, there is no check to enforce that the new address is not `address(0)`. Setting the new address to `address(0)` would "pause" `poolMint` and `poolBurn`, which is the role of a `PAUSER`, not `DEFAULT_ADMIN`.

Code corrected:

`poolAddress` state variable as well as all the functions which modify it have been removed.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Freeze User Frontrunning

Note Version 1

A misbehaving user could front-run the transaction which includes their address to the frozen ones.

7.2 BURNER_ROLE Cannot Burn From Frozen User

Note Version 2

Since `_burn` triggers `_beforeTokenTransfer`, the burner cannot burn the tokens from frozen users. That would prevent the burner to "slash" misbehaving users. However, the burner could still be able to perform the slashing by batching two transaction, one to unfreeze the user and one to burn their tokens.