



Trains, Contract, Code Review and Security Analysis Report

Customer: Trains
Prepared on: 27th October 2023
Platform: Binance
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer	1
Document	2
Introduction	3
Project Scope	5
Executive Summary	6
Code Quality	7
Documentation	8
Use of Dependencies	9
AS-IS Overview	9
Code Flow Diagram - Card.sol	13
Pool.sol	13
Trains.sol	13
Code Flow Diagram - Slither Results Log	13
Audit Findings	33
Discussion	33
Conclusion	34
Note For Contract Users	35
Our Methodology	38
Disclaimers	41

Disclaimer

This document may contain confidential information about its systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities are fixed - upon the decision of the customer.

Document

Name	Smart Contract Code Review and Security Analysis Report of Trains
Platform	Binance/ Solidity
File	Card.sol
MD5 hash	56f9d0ac48c0d16d07c16633dd2eb29f
SHA256 hash	f5f63426b6e22f24e8b80fa419cb06e9602e1697899ca50fca0f17dd78f8f3db
File	Mine.sol
MD5 hash	1fc3918b71e1e1ca7a893b25ca7c3a34
SHA256 hash	adae385ed178b6bf8a9105b3186079d6fe163c4b8145f2380e71e1ea9cc7f0ea
File	Pool.sol
MD5 hash	49e4e0fdef008744966e4a133a7e1c25
SHA256 hash	7dd9e12a4e882857bac6b2e074974f44ab8147adb12671d1ecd3d7eff0458e5a
File	Trains.sol

MD5 hash	dd6e21d943e1905fe3697f2328d9d587
SHA256 hash	68b68bee1ec6ded4e0f14a70b2a7f16146d61d5f934d2aac57a157d9d2b60d55
Date	27/10/2023

Introduction

RD Auditors (Consultant) were contracted by trains (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report represents the findings of the security assessment of the customer's smart contract and its code review conducted between 24th - 27th October 2023.

This contract consists of four files.

Project Scope


The scope of the project is a smart contract. We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to):


- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary


According to the assessment, the customer's solidity smart contract is now **well-Secured**.

You are Here

 Insecure






 Poorly Secured

 Secure

 Well-Secured

Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, the manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found the following;

Total Issues	0
 Critical	0
 High	0
 Medium	0
 Low	0
 Very Low	0

Code Quality

The libraries within this smart contract are part of a logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned to a specific address and its properties/methods can be reused many times by other contracts.

The Trains team has not provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Overall, the code is not commented. Commenting can provide rich documentation for functions, return variables and more. Use of the Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given the Trains code as a github link:

<https://github.com/trainsAdmin/trains-contract/tree/main/contracts>

The hash of that file is mentioned in the table. As mentioned above, it's recommended to write comments on smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects and even core code blocks that are written well and systematically.

AS-IS Overview

Trains .sol

File And Function Level Report

Contract: Card

Inherit: AccessControl, Pausable, ERC1155, ERC1155Burnable, ERC1155Supply, ERC1155UPISStorage

Import: AccessControl, Pausable, ERC1155, ERC1155Burnable, ERC1155Supply, ERC1155UPISStorage

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	SupportsInterface	read	Passed	All Passed	No Issue	Passed
2	Uri	read	Passed	All Passed	No Issue	Passed
3	SetBaseURI	onlyRole	Passed	All Passed	No Issue	Passed
4	pause	onlyRole	Passed	All Passed	No Issue	Passed
5	Unpause	onlyRole	Passed	All Passed	No Issue	Passed
6	mint	onlyRole	Passed	All Passed	No Issue	Passed
7	mintBatch	onlyRole	Passed	All Passed	No Issue	Passed
8	beforeTokenTransfer	internal	Passed	All Passed	No Issue	Passed

Contract: Mine

Inherit: AccessControlUpgradeable, ReentrancyGuardUpgradeable, ERC1155HolderUpgradeable

Import: ReentrancyGuardUpgradeable, AccessControlUpgradeable, IERC1155Upgradeable, ERC1155HolderUpgradeable, SafeERC20Upgradeable, CountersUpgradeable

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	initialize	write	Passed	All Passed	No Issue	Passed
2	SupportsInterface	read	Passed	All Passed	No Issue	Passed
3	SetPayee	onlyRole	Passed	All Passed	No Issue	Passed
4	SetWithdrawAuditLimit	onlyRole	Passed	All Passed	No Issue	Passed
5	getWithdrawList	onlyRole	Passed	All Passed	No Issue	Passed
6	rewardPerHashRate	read	Passed	All Passed	No Issue	Passed
7	earned	read	Passed	All Passed	No Issue	Passed
8	Stake	write	Passed	All Passed	No Issue	Passed
9	UnStake	write	Passed	All Passed	No Issue	Passed
10	Claim	external	Passed	All Passed	No Issue	Passed
11	auditWithdraw	onlyRole	Passed	All Passed	No Issue	Passed

Contract: Pool

Inherit: AccessControlUpgradeable, ReentrancyGuardUpgradeable

Import: AccessControlUpgradeable, IERC20Upgradeable,
ReentrancyGuardUpgradeable, ReentrancyGuard,
safeERC20Upgradeable

Observation: Passed

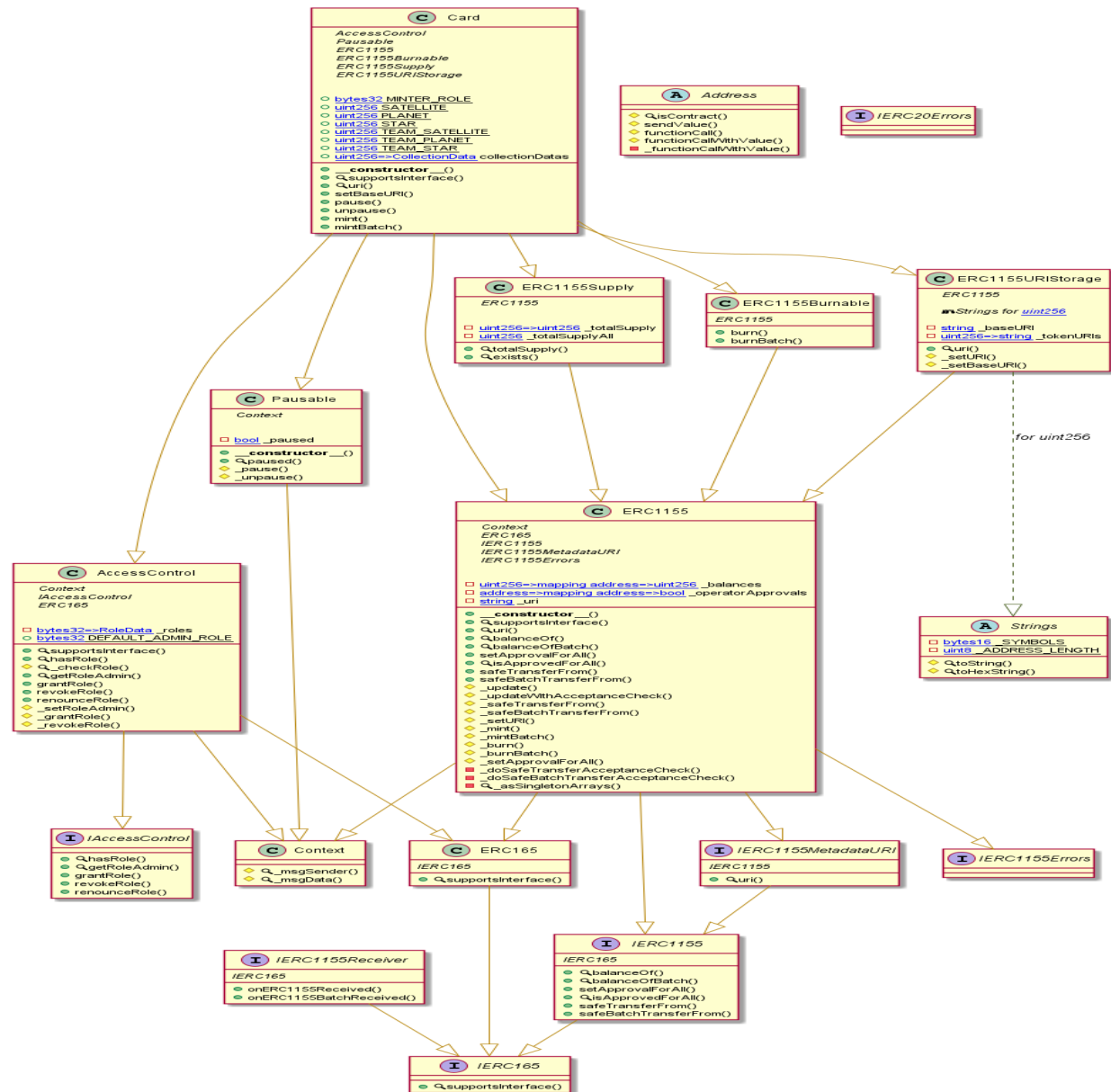
Test Report: Passed

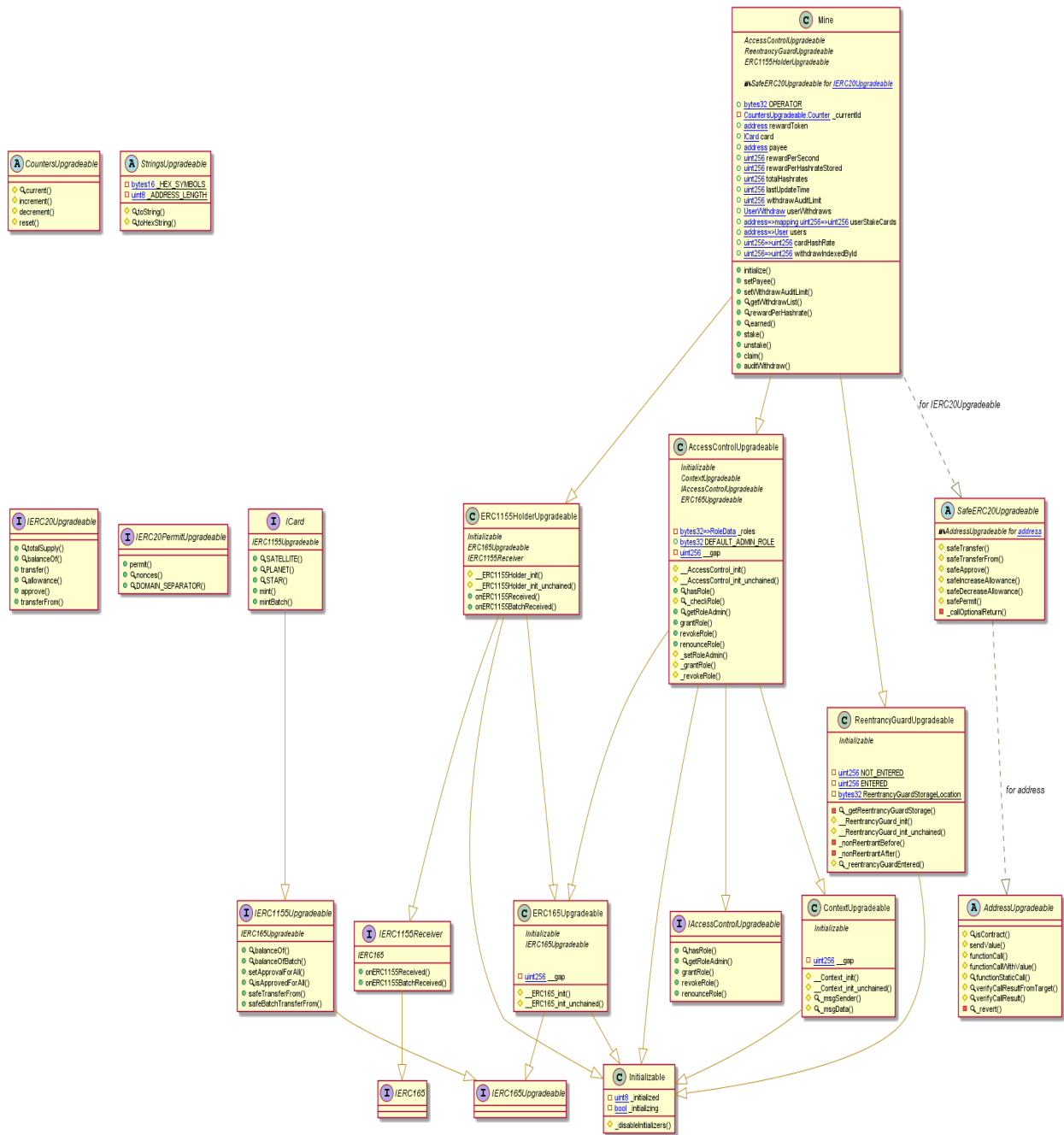
Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	initialize	write	Passed	All Passed	No Issue	Passed
2	setPayee	write	Passed	All Passed	No Issue	Passed
3	setMinStaking	write	Passed	All Passed	No Issue	Passed
4	setMaxStaking	onlyOperator	Passed	All Passed	No Issue	Passed
5	setTotalSupply	onlyOperator	Passed	All Passed	No Issue	Passed
6	hindSuperior	onlyOperator	Passed	All Passed	No Issue	Passed
7	depositToken	write	Passed	All Passed	No Issue	Passed
8	depositNFT	write	Passed	All Passed	No Issue	Passed
9	_deposit	write	Passed	All Passed	No Issue	Passed
10	_checkUserDeposit	read	Passed	All Passed	No Issue	Passed

Contract: Trains
Inherit: AccessControl, Pausable, ERC20, ERC20Burnable
Import: AccessControl, Pausable, ERC20, ERC20Burnable
Observation: Passed
Test Report: Passed

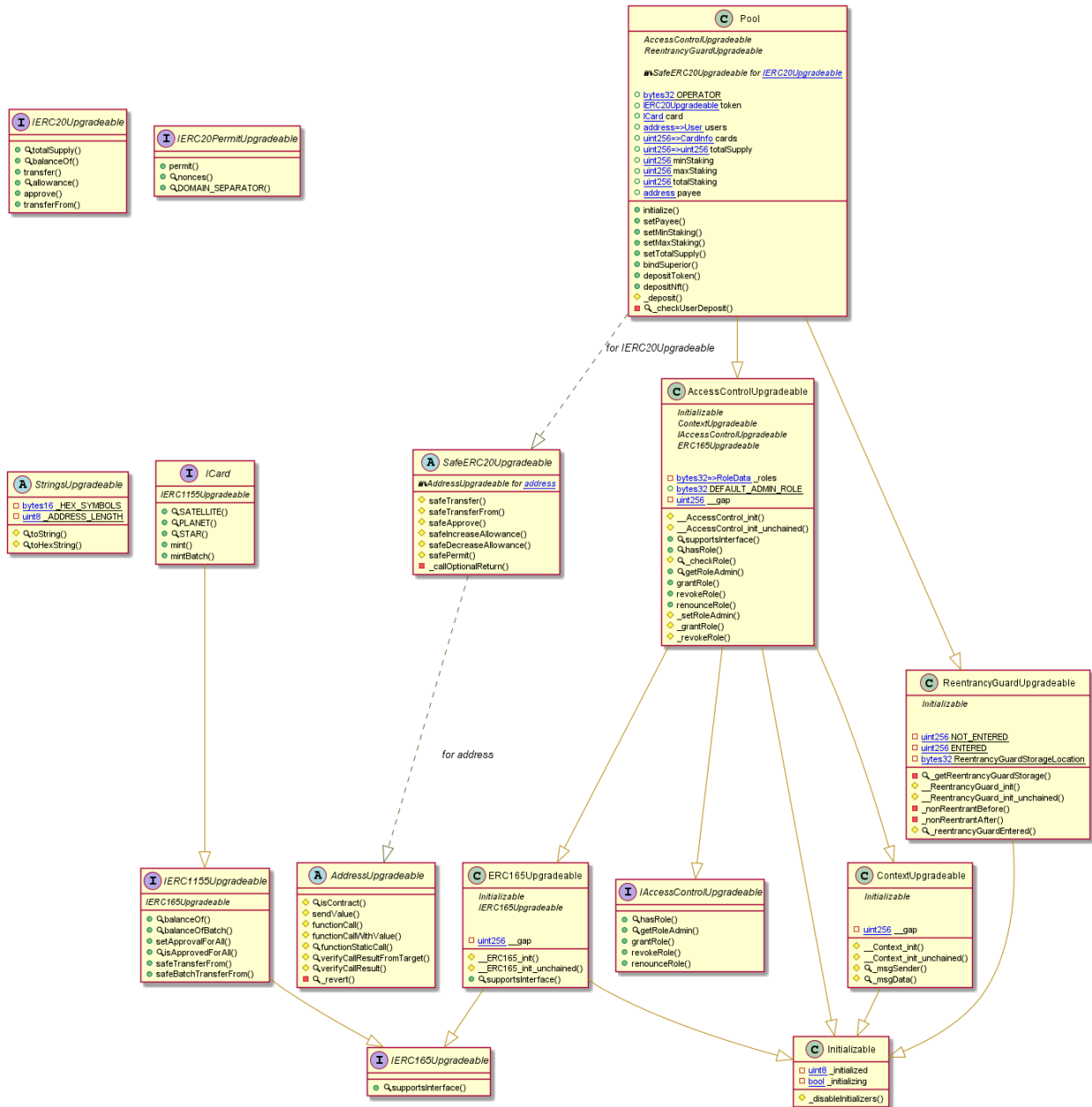
Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Pause	write	Passed	All Passed	No Issue	Passed
2	Unpause	write	Passed	All Passed	No Issue	Passed
3	mint	write	Passed	All Passed	No Issue	Passed
4	_beforeTokenTransfer	write	Passed	All Passed	No Issue	Passed

File: Mine.sol

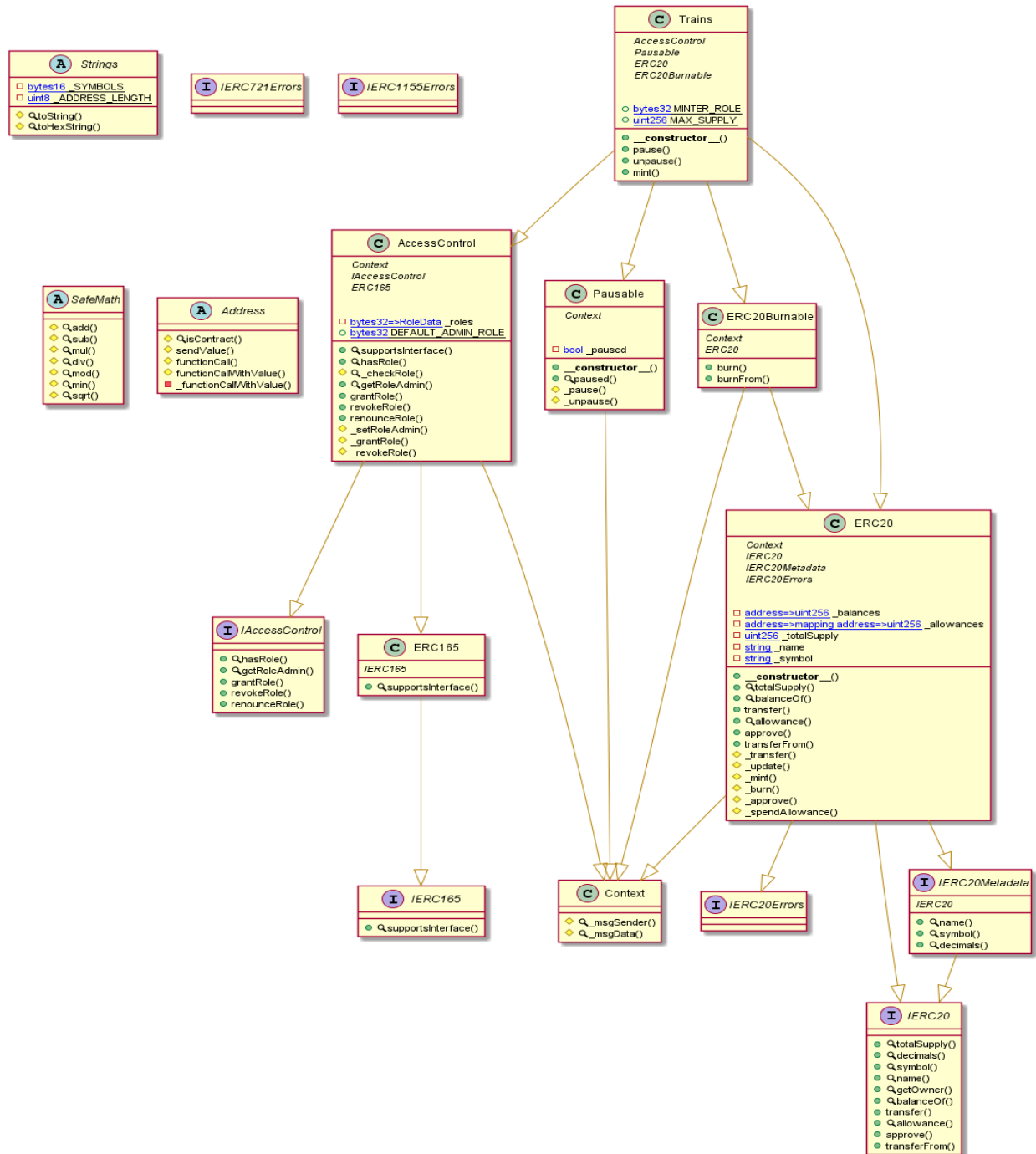




Pool.sol



Trains.sol



Code Flow Diagram - Slither Results Log

Card.sol

Gas costs:

Gas requirement of function Card.pause is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 969:4:

Gas costs:

Gas requirement of function Card.unpause is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 973:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

[more](#)

Pos: 520:8:

Similar variable names:

Card.mint(address,uint256,uint256,bytes) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.

Pos: 983:27:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

[more](#)

Pos: 297:8:

Mine.sol

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in
Mine.initialize(address,address,address): Could potentially lead to re-entrancy vulnerability.
Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 894:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Mine.unstake(uint256[],uint256[]): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 1008:4:

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

[more](#)

Pos: 611:8:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

[more](#)

Pos: 1046:32:

Gas costs:

Gas requirement of function Mine.stake is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 975:4:

Gas costs:

Gas requirement of function Mine.unstake is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 1008:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

[more](#)

Pos: 1014:8:

Constant/View/Pure functions:

Mine.getWithdrawList(uint256,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 937:4:

Similar variable names:

Mine.auditWithdraw(uint256,bool) : Variables have very similar names "userWithdraw" and "userWithdraws". Note: Modifiers are currently not considered by this static analysis.

Pos: 1070:16:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

[more](#)

Pos: 1016:12:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.

[more](#)

Pos: 1089:8:

Data truncated:

Division of integer values yields an integer value again. That means e.g. $10 / 100 = 0$ instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 969:13:

Pool.sol

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in
Pool.depositNft(uint256,uint256,uint8): Could potentially lead to re-entrancy
vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 105:4:

Gas costs:

Gas requirement of function Pool.initialize is infinite: If the gas requirement of a
function is higher than the block gas limit, it cannot be executed. Please avoid
loops in your functions or actions that modify large areas of storage (this
includes clearing or copying arrays in storage)

Pos: 50:4:

Gas costs:

Gas requirement of function Pool.depositNft is infinite: If the gas requirement of a
function is higher than the block gas limit, it cannot be executed. Please avoid
loops in your functions or actions that modify large areas of storage (this
includes clearing or copying arrays in storage)

Pos: 105:4:

Similar variable names:

Pool.initialize(address,address,address) : Variables have very similar names "token" and "token_". Note: Modifiers are currently not considered by this static analysis.

Pos: 54:34:

Similar variable names:

Pool._checkUserDeposit(uint256,uint256,uint8) : Variables have very similar names "minStaking" and "maxStaking". Note: Modifiers are currently not considered by this static analysis.

Pos: 128:50:

No return:

ICard.STAR(): Defines a return type but never explicitly returns a value.

Pos: 9:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

[more](#)

Pos: 129:8:

Trains.sol

Gas costs:

Gas requirement of function `Trains.pause` is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 20:22:

Gas costs:

Gas requirement of function `Trains.unpause` is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 24:22:

Gas costs:

Gas requirement of function `Trains.mint` is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 28:22:

Constant/View/Pure functions:

`Trains._beforeTokenTransfer(address,address,uint256)` : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 33:22:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

[more](#)

Pos: 29:26:

Solidity Static Analysis

Card.sol

```
ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes) (Card.sol#720-746) ignores return value by IERC1155Receiver(to).onERC1155Received(operator,from,id,value,data) (Card.sol#729-744)
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes) (Card.sol#752-780) ignores return value by IERC1155Receiver(to).onERC1155BatchReceived(operator,from,ids,values,data) (Card.sol#761-778)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

```
Variable 'ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes).response (Card.sol#729)' in ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes) (Card.sol#720-746) potentially used before declaration: response != IERC1155Receiver.onERC1155Received.selector (Card.sol#730)
Variable 'ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes).reason (Card.sol#734)' in ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes) (Card.sol#720-746) potentially used before declaration: reason.length == 0 (Card.sol#735)
Variable 'ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes).reason (Card.sol#734)' in ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes) (Card.sol#752-780) potentially used before declaration: revert(uint256,uint256)(32 + reason,mload(uint256)(reason)) (Card.sol#741)
Variable 'ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes).response (Card.sol#762)' in ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes) (Card.sol#752-780) potentially used before declaration: response != IERC1155Receiver.onERC1155BatchReceived.selector (Card.sol#764)
Variable 'ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes).reason (Card.sol#768)' in ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes) (Card.sol#752-780) potentially used before declaration: reason.length == 0 (Card.sol#769)
Variable 'ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes).reason (Card.sol#768)' in ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes) (Card.sol#752-780) potentially used before declaration: revert(uint256,uint256)(32 + reason,mload(uint256)(reason)) (Card.sol#775)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
```

```
Strings.toString(uint256) (Card.sol#9-27) uses assembly
- INLINE ASM (Card.sol#14-16)
- INLINE ASM (Card.sol#19-21)
Address.isContract(address) (Card.sol#245-252) uses assembly
- INLINE ASM (Card.sol#248-250)
Address.functionCallWithValue(address,bytes,uint256,string) (Card.sol#291-313) uses assembly
- INLINE ASM (Card.sol#305-308)
ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes) (Card.sol#720-746) uses assembly
- INLINE ASM (Card.sol#740-742)
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes) (Card.sol#752-780) uses assembly
- INLINE ASM (Card.sol#774-776)
ERC1155.asSingletonArrays(uint256,uint256) (Card.sol#785-806) uses assembly
- INLINE ASM (Card.sol#790-805)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
AccessControl.setRoleAdmin(bytes32,bytes32) (Card.sol#141-145) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (Card.sol#291-313) is never used and should be removed
Address.functionCall(address,bytes) (Card.sol#261-263) is never used and should be removed
Address.functionCall(address,bytes,string) (Card.sol#265-271) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (Card.sol#273-279) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (Card.sol#281-289) is never used and should be removed
Address.isContract(address) (Card.sol#245-252) is never used and should be removed
Address.sendValue(address,uint256) (Card.sol#254-259) is never used and should be removed
Context._msgData() (Card.sol#78-80) is never used and should be removed
ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes) (Card.sol#752-780) is never used and should be removed
ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes) (Card.sol#720-746) is never used and should be removed
ERC1155.URIStorage.setURI(uint256,string) (Card.sol#895-898) is never used and should be removed
Strings.toHexString(address) (Card.sol#47-49) is never used and should be removed
Strings.toHexString(uint256) (Card.sol#29-33) is never used and should be removed
Strings.toHexString(uint256,uint256) (Card.sol#35-45) is never used and should be removed
Strings.toString(uint256) (Card.sol#9-27) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```

Pragma version^0.8.19 (Card.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.16
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (Card.sol#254-259):
- (success) = recipient.call{value: amount}() (Card.sol#257)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (Card.sol#291-313):
- (success,returndata) = target.call{value: weiValue}(data) (Card.sol#299)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
Card.sol analyzed (18 contracts with 84 detectors), 34 result(s) found

```

Mine.sol

```

Reentrancy in Mine.auditWithdraw(uint256,bool) (Mine.sol#839-866):
  External calls:
  - IERC20Upgradeable(rewardToken).safeTransfer(userWithdraw.account,userWithdraw.amount) (Mine.sol#848-851)
  State variables written after the call(s):
  - userWithdraws[index - 1] = userWithdraws[userWithdraws.length - 1] (Mine.sol#858)
  Mine.userWithdraws (Mine.sol#637) can be used in cross function reentrancies:
  - Mine.auditWithdraw(uint256,bool) (Mine.sol#839-866)
  - Mine.getWithdrawList(uint256,uint256) (Mine.sol#711-727)
  - Mine.userWithdraws (Mine.sol#637)
  - userWithdraws.pop() (Mine.sol#862)
  Mine.userWithdraws (Mine.sol#637) can be used in cross function reentrancies:
  - Mine.auditWithdraw(uint256,bool) (Mine.sol#839-866)
  - Mine.getWithdrawList(uint256,uint256) (Mine.sol#711-727)
  - Mine.userWithdraws (Mine.sol#637)
  - withdrawIndexedById[userWithdraws[index - 1].id] = index (Mine.sol#859)
  Mine.withdrawIndexedById (Mine.sol#659) can be used in cross function reentrancies:
  - Mine.auditWithdraw(uint256,bool) (Mine.sol#839-866)
  - Mine.withdrawIndexedById (Mine.sol#659)
  - delete withdrawIndexedById[id] (Mine.sol#863)
  Mine.withdrawIndexedById (Mine.sol#659) can be used in cross function reentrancies:
  - Mine.auditWithdraw(uint256,bool) (Mine.sol#839-866)
  - Mine.withdrawIndexedById (Mine.sol#659)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

Mine.claim().userWithdraw (Mine.sol#826) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

Mine.initialize(address,address).rewardToken (Mine.sol#669) lacks a zero-check on :
  - rewardToken = _rewardToken (Mine.sol#673)
Mine.initialize(address,address,address).payee (Mine.sol#671) lacks a zero-check on :
  - payee = _payee (Mine.sol#675)
Mine.setPayee(address).newPayee (Mine.sol#701) lacks a zero-check on :
  - payee = newPayee (Mine.sol#703)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Mine.stake(uint256[],uint256[]) (Mine.sol#749-780) has external calls inside a loop: require(bool,string)(ids[i] == card.SATELLITE() || ids[i] == card.PLANET() || ids[i] == card.STAR(),card id error) (Mine.sol#757)
Mine.unstake(uint256[],uint256[]) (Mine.sol#782-812) has external calls inside a loop: require(bool,string)(ids[i] == card.SATELLITE() || ids[i] == card.PLANET() || ids[i] == card.STAR(),card id error) (Mine.sol#789)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#calls-inside-a-loop

Reentrancy in Mine.auditWithdraw(uint256,bool) (Mine.sol#839-866):
  External calls:
  - IERC20Upgradeable(rewardToken).safeTransfer(userWithdraw.account,userWithdraw.amount) (Mine.sol#848-851)
  Event emitted after the call(s):
  - Withdrawn(userWithdraw.account,id,pass,userWithdraw.amount) (Mine.sol#865)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Mine.claim() (Mine.sol#814-837) uses timestamp for comparisons
  Dangerous comparisons:
  - require(bool,string)(user.reward > 0,no reward) (Mine.sol#816)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

AddressUpgradeable._revert(bytes,string) (Mine.sol#244-253) uses assembly
  - INLINE ASM (Mine.sol#246-249)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

```

```

AccessControlUpgradeable.__AccessControl_init() (Mine.sol#538-539) is never used and should be removed
AccessControlUpgradeable.__AccessControl_init_unchained() (Mine.sol#541-542) is never used and should be removed
AccessControlUpgradeable.__setRoleAdmin(bytes32,bytes32) (Mine.sol#591-595) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes) (Mine.sol#172-174) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (Mine.sol#184-190) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes) (Mine.sol#203-205) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (Mine.sol#207-214) is never used and should be removed
AddressUpgradeable.sendValue(address,uint256) (Mine.sol#165-170) is never used and should be removed
AddressUpgradeable.verifyCallResult(bool,bytes,string) (Mine.sol#232-242) is never used and should be removed
ContextUpgradeable.__Context_init() (Mine.sol#324-325) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (Mine.sol#327-328) is never used and should be removed
ContextUpgradeable.__msgData() (Mine.sol#333-335) is never used and should be removed
CountersUpgradeable.decrement(CountersUpgradeable.Counter) (Mine.sol#70-76) is never used and should be removed
CountersUpgradeable.reset(CountersUpgradeable.Counter) (Mine.sol#78-80) is never used and should be removed
ERC1155HolderUpgradeable.__ERC1155Holder_init() (Mine.sol#353-354) is never used and should be removed
ERC1155HolderUpgradeable.__ERC1155Holder_init_unchained() (Mine.sol#356-357) is never used and should be removed
ERC165Upgradeable.__ERC165_init() (Mine.sol#342-343) is never used and should be removed
ERC165Upgradeable.__ERC165_init_unchained() (Mine.sol#345-346) is never used and should be removed
Initializable.disableInitializers() (Mine.sol#314-320) is never used and should be removed
ReentrancyGuardUpgradeable.__ReentrancyGuard_init() (Mine.sol#511-513) is never used and should be removed
ReentrancyGuardUpgradeable.__ReentrancyGuard_init_unchained() (Mine.sol#515-518) is never used and should be removed
ReentrancyGuardUpgradeable.__reentrancyGuardEntered() (Mine.sol#532-533) is never used and should be removed
SafeERC20Upgradeable.safeApprove(IERC20Upgradeable,address,uint256) (Mine.sol#437-447) is never used and should be removed
SafeERC20Upgradeable.safeDecreaseAllowance(IERC20Upgradeable,address,uint256) (Mine.sol#458-469) is never used and should be removed
SafeERC20Upgradeable.safeIncreaseAllowance(IERC20Upgradeable,address,uint256) (Mine.sol#449-456) is never used and should be removed
SafeERC20Upgradeable.safePermit(IERC20PermitUpgradeable,address,address,uint256,uint256,uint8,bytes32,bytes32) (Mine.sol#471-485) is never used and should be removed
SafeERC20Upgradeable.safeTransferFrom(IERC20Upgradeable,address,address,uint256) (Mine.sol#428-435) is never used and should be removed
StringsUpgradeable.toHexString(address) (Mine.sol#154-156) is never used and should be removed
StringsUpgradeable.toHexString(uint256) (Mine.sol#129-140) is never used and should be removed
StringsUpgradeable.toHexString(uint256,uint256) (Mine.sol#142-152) is never used and should be removed
StringsUpgradeable.toString(uint256) (Mine.sol#109-127) is never used and should be removed

```

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code>

Pragma version^0.8.19 (Mine.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.16
solc-0.8.19 is not recommended for deployment

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity>

```

Low level call in AddressUpgradeable.sendValue(address,uint256) (Mine.sol#165-170):
- (success) = recipient.call{value: amount}() (Mine.sol#168)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (Mine.sol#192-201):
- (success,returndata) = target.call{value: value}(data) (Mine.sol#199)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (Mine.sol#207-214):
- (success,returndata) = target.staticcall(data) (Mine.sol#212)

```

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls>

```

Function ICard.SATELLITE() (Mine.sol#48) is not in mixedCase
Function ICard.PLANET() (Mine.sol#49) is not in mixedCase
Function ICard.STAR() (Mine.sol#50) is not in mixedCase
Function ContextUpgradeable.__Context_init() (Mine.sol#324-325) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (Mine.sol#327-328) is not in mixedCase
Variable ContextUpgradeable.__gap (Mine.sol#337) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init() (Mine.sol#342-343) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init_unchained() (Mine.sol#345-346) is not in mixedCase
Variable ERC165Upgradeable.__gap (Mine.sol#348) is not in mixedCase
Function ERC1155HolderUpgradeable.__ERC1155Holder_init() (Mine.sol#353-354) is not in mixedCase
Function ERC1155HolderUpgradeable.__ERC1155Holder_init_unchained() (Mine.sol#356-357) is not in mixedCase
Function IERC20PermitUpgradeable.DOMAIN_SEPARATOR() (Mine.sol#415) is not in mixedCase
Function ReentrancyGuardUpgradeable.__ReentrancyGuard_init() (Mine.sol#511-513) is not in mixedCase
Function ReentrancyGuardUpgradeable.__ReentrancyGuard_init_unchained() (Mine.sol#515-518) is not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init() (Mine.sol#538-539) is not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init_unchained() (Mine.sol#541-542) is not in mixedCase
Variable AccessControlUpgradeable.__gap (Mine.sol#611) is not in mixedCase
Parameter Mine.initialize(address,address,address)._rewardToken (Mine.sol#669) is not in mixedCase
Parameter Mine.initialize(address,address,address)._card (Mine.sol#670) is not in mixedCase
Parameter Mine.initialize(address,address,address)._payee (Mine.sol#671) is not in mixedCase
Parameter Mine.setWithdrawAuditLimit(uint256)._withdrawAuditLimit (Mine.sol#706) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
Mine.sol analyzed (19 contracts with 84 detectors), 67 result(s) found

```

Pool.sol

```
Reentrancy in Pool.depositNft(uint256,uint256,uint8) (Pool.sol#635-648):
  External calls:
    - token.safeTransferFrom(_msgSender(),payee,price) (Pool.sol#645)
    - card.mint(msg.sender,id,amount,) (Pool.sol#646)
  State variables written after the call(s):
    - _deposit(price,period,id,renewal) (Pool.sol#647)
    - users[_msgSender()].stakingTotal += amount (Pool.sol#652)
  Pool.users (Pool.sol#558) can be used in cross function reentrancies:
    - Pool.users (Pool.sol#558)
Reentrancy in Pool.depositToken(uint256,uint256,uint8) (Pool.sol#629-633):
  External calls:
    - token.safeTransferFrom(_msgSender(),payee,amount) (Pool.sol#631)
  State variables written after the call(s):
    - _deposit(amount,period,0,renewal) (Pool.sol#632)
    - users[_msgSender()].stakingTotal += amount (Pool.sol#652)
  Pool.users (Pool.sol#558) can be used in cross function reentrancies:
    - Pool.users (Pool.sol#558)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

Pool.initialize(address,address,address).payee_ (Pool.sol#580) lacks a zero-check on :
  - payee = payee_ (Pool.sol#586)
Pool.setPayee(address).newPayee (Pool.sol#600) lacks a zero-check on :
  - payee = newPayee (Pool.sol#602)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

```
Reentrancy in Pool.depositNft(uint256,uint256,uint8) (Pool.sol#635-648):
  External calls:
    - token.safeTransferFrom(_msgSender(),payee,price) (Pool.sol#645)
    - card.mint(msg.sender,id,amount,) (Pool.sol#646)
  State variables written after the call(s):
    - _deposit(price,period,id,renewal) (Pool.sol#647)
    - totalStaking += amount (Pool.sol#651)
Reentrancy in Pool.depositToken(uint256,uint256,uint8) (Pool.sol#629-633):
  External calls:
    - token.safeTransferFrom(_msgSender(),payee,amount) (Pool.sol#631)
  State variables written after the call(s):
    - _deposit(amount,period,0,renewal) (Pool.sol#632)
    - totalStaking += amount (Pool.sol#651)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

AddressUpgradeable._revert(bytes,string) (Pool.sol#142-151) uses assembly
  - INLINE ASM (Pool.sol#144-147)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```


Page No : 31

Trains.sol

```
Strings.toString(uint256) (Trains.sol#8-24) uses assembly
- INLINE ASM (Trains.sol#11-13)
- INLINE ASM (Trains.sol#16-18)
Address.isContract(address) (Trains.sol#342-349) uses assembly
- INLINE ASM (Trains.sol#345-347)
Address.functionCallWithValue(address,bytes,uint256,string) (Trains.sol#388-410) uses assembly
- INLINE ASM (Trains.sol#402-405)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

AccessControl.setRoleAdmin(bytes32,bytes32) (Trains.sol#138-142) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (Trains.sol#388-410) is never used and should be removed
Address.functionCall(address,bytes) (Trains.sol#358-360) is never used and should be removed
Address.functionCall(address,bytes,string) (Trains.sol#362-368) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (Trains.sol#370-376) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (Trains.sol#378-386) is never used and should be removed
Address.isContract(address) (Trains.sol#342-349) is never used and should be removed
Address.sendValue(address,uint256) (Trains.sol#351-356) is never used and should be removed
Context.msgData() (Trains.sol#75-77) is never used and should be removed
SafeMath.add(uint256,uint256) (Trains.sol#263-268) is never used and should be removed
SafeMath.div(uint256,uint256) (Trains.sol#296-298) is never used and should be removed
SafeMath.div(uint256,uint256,string) (Trains.sol#300-309) is never used and should be removed
SafeMath.min(uint256,uint256) (Trains.sol#324-326) is never used and should be removed
SafeMath.mod(uint256,uint256) (Trains.sol#311-313) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (Trains.sol#315-322) is never used and should be removed
SafeMath.mul(uint256,uint256) (Trains.sol#285-294) is never used and should be removed
SafeMath.sqrt(uint256) (Trains.sol#328-339) is never used and should be removed
SafeMath.sub(uint256,uint256) (Trains.sol#270-272) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (Trains.sol#274-283) is never used and should be removed
Strings.toHexString(address) (Trains.sol#44-46) is never used and should be removed
Strings.toHexString(uint256) (Trains.sol#26-30) is never used and should be removed
Strings.toHexString(uint256,uint256) (Trains.sol#32-42) is never used and should be removed
Strings.toString(uint256) (Trains.sol#8-24) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
Pragma version^0.8.19 (Trains.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.16
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (Trains.sol#351-356):
- (success) = recipient.call{value: amount}() (Trains.sol#354)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (Trains.sol#388-410):
- (success,returndata) = target.call{value: weiValue}(data) (Trains.sol#396)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Trains (Trains.sol#745-770) does not implement functions:
- IERC20Metadata.decimals() (Trains.sol#458)
- IERC20.getOwner() (Trains.sol#421)
- IERC20Metadata.name() (Trains.sol#448)
- IERC20Metadata.symbol() (Trains.sol#453)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions
Trains.sol analyzed (17 contracts with 84 detectors), 31 result(s) found
```

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
High	High level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial functions.
Medium	Medium level vulnerabilities are important to fix; however, they cannot lead to lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest Code Style/ Best Practice	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored.

Audit Findings

Critical:

No critical severity vulnerabilities were found.

High:

No high severity vulnerabilities were found.

Medium:

No medium severity vulnerabilities were found.

Low:

No low severity vulnerabilities were found.

Very Low:

No very low severity vulnerabilities were found.

Discussion

Version Pragma: The Solidity version pragma is set to ^0.8.19. The caret (^) symbol should be used with caution, as it allows the compiler to use any version greater than 0.8.19. To minimize the risk of breaking changes in future compiler versions, consider specifying an exact version

```
1    // SPDX-License-Identifier: MIT
2    pragma solidity ^0.8.19;
3
```

Conclusion

We were given a contract file and have used all possible tests based on the given object. So it is now ready for mainnet deployment. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

The security state of the reviewed contract is **“well-Secured”**.

Note For Contract Users

There are several administrator functions. Those can be called by the administrator's wallet only. So, if the administrator's wallet is compromised, then it carries the risk of the contract becoming vulnerable.

SetBaseURI: The 'setBaseURI' function can only be called by an account that has the `DEFAULT_ADMIN_ROLE`. If the sender of the transaction does not have that role, the function call will fail with an error.

Pause: The 'pause' function is restricted to be called only by an account with the `DEFAULT_ADMIN_ROLE`. This ensures that only an authorized administrator can pause the contract.

UnPause: The 'unpause' function is also restricted to be called only by an account with the `DEFAULT_ADMIN_ROLE`, ensuring that only authorized administrators can unpause the contract.

Mint: This ensures that only accounts with the '`MINTER_ROLE`' can call the mint function.

MintBatch: The function is restricted by the `onlyRole(MINTER_ROLE)` modifier, which ensures that only accounts with the `MINTER_ROLE` can call this function. The `MINTER_ROLE` is a role defined in the contract, typically used to control who has permission to mint new tokens.

SetPayee: The 'setPayee' function is restricted to the admin. In this function, only the account with the DEFAULT_ADMIN_ROLE can change the payee address.

SetWithdrawAudit: This function is restricted by the onlyRole (DEFAULT_ADMIN_ROLE) modifier. This means that only the account that was initially granted the DEFAULT_ADMIN_ROLE (the admin) can call these functions and change certain contract settings.

auditWithdraw: The auditWithdraw function is restricted to be called only by accounts with the OPERATOR role. This role is established in the contract and is meant to represent operators or administrators with the authority to audit withdrawal requests. The auditWithdraw function allows authorized operators to review and approve or deny withdrawal requests, transferring the funds to the user or returning them to the user's reward balance based on the audit outcome.

SetMinStaking/SetMaxStaking/SetTotalSupply: setMinStaking, setMaxStaking, and setTotalSupply functions are administrative functions that allow authorized administrators to configure parameters related to staking and total token supply in the contract. These functions are often used to control and adjust the contract's behavior and settings as needed.

administrator has full control over the smart contract. Thus, technical auditing does not guarantee the project's ethical side.

Please do your due diligence before investing. Our audit report is never an investment advice.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyse the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

RD Auditors Disclaimer

The smart contracts given for audit have been analysed in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Because the total number of test cases are unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.



Email: info@rdauditors.com

Website: www.rdauditors.com

