CERTIK

# CertiK Final Audit Report for The Sandbox

# Contents

## 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 Verification Services Agreement between CertiK and Sandbox (the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, 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.

## About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, CertiK's mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that projects are checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and assessments to each project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance's BGBP and Paxos Gold to decentralized oracles such as Band Protocol and Tellor. CertiK customizes its engineering tool kits, while applying cutting-edge research on smart contracts, for each client on its project to offer a high quality deliverable.  For more information: https://certik.io.

## Executive Summary

This report has been prepared for **Sandbox** to discover issues and vulnerabilities in the source code of their **Catalyst & ERC-20 Wrapper Smart Contracts** as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, 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.

# Testing Summary

SECURITY LEVEL


Smart Contract Audit

This report has been prepared as a product of the Smart Contract Audit request by Sandbox.

This audit was conducted to discover issues and vulnerabilities in the source code of Sandbox's Catalyst & ERC Wrapper Contracts.

| | |
|---|---|
| TYPE | Smart Contracts & Token Wrappers |
| SOURCE CODE | https://github.com/thesandboxgame/sandbox-private-contracts/ |
| PLATFORM | EVM |
| LANGUAGE | Solidity |
| REQUEST DATE | June 25, 2020 |
| FINAL DELIVERY DATE | August 19, 2020 |
| METHODS | A comprehensive examination has been performed using Dynamic Analysis, Static Analysis, and Manual Review. |

# Review Notes

## Introduction

CertiK team was contracted by the Sandbox team to audit the design and implementations of their Catalyst & ERC-20 Wrapper smart contracts that are meant to be utilized within their on-chain gaming platform that utilizes user-generated content to empower creatives in monetizing their 3D voxel creations on the platform and share them with users across the world.

The audited source code links are:

- Catalyst & ERC-20 Wrappers: https://github.com/thesandboxgame/sandbox-private-contracts/tree/audit_20200625

The goal of this audit was to review the Solidity implementation for its business model, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

The findings of the initial audit have been conveyed to the team behind the contract implementations and the source code is expected to be re-evaluated before another round of auditing has been carried out.

The development team of Sandbox dealt with the preliminary Exhibits of the report in PR #33 which was merged on the tree below. Each “Alleviation” chapter of the Exhibits is meant to describe the actions that were taken in the following version:

- Catalyst & ERC-20 Wrappers: https://github.com/thesandboxgame/sandbox-private-contracts/tree/a026cb7906e47e8b085e362b3f7c0e59ffb18cc1

A third round of audit proceeded were we provided our new findings to the Sandbox team by assessing the following version:

- Catalyst & ERC-20 Wrappers: https://github.com/thesandboxgame/sandbox-private-contracts/tree/audit_catalyst_20200720

Finally, a fourth round of audit was carried out on the following and final link provided to us by Sandbox:

- Catalyst & ERC-20 Wrappers: https://github.com/thesandboxgame/sandbox-private-contracts/tree/audit_catalyst_20200723

## Documentation

The sources of truth regarding the operation of the contracts in scope are **something we would advise to be expanded**. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions as well as the relevant markdown documentation.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the Sandbox team or reported an issue.

## Summary

The codebase of the project, especially with regards to the Catalyst concept, attempts to fulfill a use case that is intricate and ambitious and as such, **inefficiencies in both the design and implementation** of the various contracts were identified and properly documented.

While **most of the issues pinpointed were of negligible importance** and mostly referred to coding standards and inefficiencies, **minor, medium flaws** were identified that should be remediated as soon as possible to ensure the contracts of the Sandbox team are of the highest standard and quality.

These inefficiencies and flaws were swiftly dealt with by the development team behind the Sandbox project and so **a second, third and fourth round of auditing proceeded**. In the meantime, a direct communication channel between us and the Sandbox team was created and maintained to aid in amending the issues identified in the report.

## Recommendations

With regards to the codebase, the main recommendation we can make is **the expansion of the documentation to address the functionalities of the contracts** from an external perspective rather than an on-code perspective. Additionally, we advise that all our findings are carefully considered and assimilated in the codebase of the project to ensure the highest code standard is achieved.

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report, enforce linters and / or coding styles as well as correct any spelling errors and mistakes that appear throughout the code **to achieve a high standard of code quality and security**.

## Conclusion

The Sandbox team dealt with the preliminary findings of our report in a very short timeframe, and we highlighted areas of our preliminary findings where fixes were applied incorrectly.

An iterative approach was utilized whereby findings were communicated to the Sandbox team in near real-time to finally reach a version where no further recommendations could be made from our end, highlighting a secure codebase.

# Findings

## Exhibit 1

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Substitution of “require” Calls w/ Modifier | Coding Style | Informational | CatalystMinter.sol: L20, L27<br>CatalystRegistry.sol: L60<br>ERC20Group.sol: L30<br>ERC20GroupGem.sol: L9<br>ERC20GroupCatalyst.sol: L10, L19, L32 |

**[INFORMATIONAL] Description:**

The “Admin.sol” contract multiple contracts of the scope inherit from provides methods via which a contract's administrator is maintained. This contract exposes an “onlyAdmin” modifier that makes sure the “msg.sender” is equal to the “_admin” of the contract, however this check is manually carried out in multiple segments of the codebase.

**Recommendations:**

These statements can be instead substituted with invocations to the “onlyAdmin” modifier. If the error message is an issue, the source of “Admin.sol” should instead be directly altered rather than manually replicating the “require” calls as they are prone to error and increase the bytecode size of the contracts.

This Exhibit is also applicable to many other “require” statements within the codebase such as those concerning "INVALID_NOT_NFT", the “_checkAuthorization” function and the “require” call

for the “_minter” role. The statement “require(msg.sender == _minter, "NOT_AUTHORIZED_MINTER")” is replicated in the codebase twice and acts as an access-control check. This should instead be converted to an “onlyMinter” modifier. Additional optimizations should occur for all other types of “require” statements that occur in the codebase.

This will, in turn, allow the functions to be exposed directly as opposed to being wrapped, as is the case with “setGemAdditionFee” internally calling “_setGemAdditionFee” after a single “require” statement for example.

**Alleviation:**

After coordination with the Sandbox team, they identified that implementing the “modifier” pattern would actually result in an increase in gas cost as the code would be substituted in-place wherever the modifier is present. While using modifiers allows “require” logic to exist within a single code block, the Sandbox team stated that they prefer using explicit “require” statements instead as per their internal coding practices.

## Exhibit 2

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Greater-than “>” Comparisons w/ Zero | Coding Style | Informational | CatalystMinter.sol: L150, L195, L267, L281, L327, L345<br>CatalystRegistry.sol: L17, L44, L45, L78, L98<br>ERC20Group.sol: L295<br>ERC20SubToken.sol: L141, L144 |

**[INFORMATIONAL] Description:**

Throughout the codebase, there are numerous instances where variables are checked using the GT operator against zero. This check costs more gas than simply conducting a not-equal operator with zero which achieves the same purpose.

**Recommendations:**

All instances of such comparisons should be swapped with a not-equal operator.

**Alleviation:**

All instances were properly swapped to utilize the more gas efficient inequality comparison.

## Exhibit 3

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Smart Contract Structure Layout | Coding Style | Informational | General |

**[INFORMATIONAL] Description:**

The structure of the codebase does not conform to the official Solidity style guide of v0.6.5. An indicative excerpt of the style guide is that functions should be grouped according to their visibility and ordered:

- constructor
- receive function (if exists)
- fallback function (if exists)
- external
- public
- internal
- private

Additionally, the internal layout of a contract should be as follows:

- Type declarations
- State variables
- Events
- Functions

However, these paradigms are not followed, increasing the illegibility of the codebase and rendering it harder to orient oneself within the codebase.

**Recommendations:**

We advise that the Solidity style guide of the version the contracts are meant to be compiled at, v0.6.5, is properly consulted to refactor the codebase according to the officially advocated structure.

**Alleviation:**

The internal styling guide of Sandbox does not conform to the above format and as such, the team understandably chose to stick to their current style guide as a shift at this point in time would require a full revamp of the existing codebase.

## Exhibit 4

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Unnecessary Named Return Variables | Ineffectual Code | Informational | CatalystMinter.sol: L88 & L91, L149 & L152, L180 & L190, L226 - L243 CatalystRegistry.sol: L12, L15, L19, L21, L88 - L90, L96, L101, L103, L180, L190, L226 - L243 ERC20Group.sol: L107, L109, L116, L118, L125, L131, L235, L236 ERC20SubToken.sol: L53, L55, L62, L72, L75, L77, L84, L87, L111, L112 |

**[INFORMATIONAL] Description:**

Within the codebase, multiple types of “return” variables are explicitly named in the function signature, however they are never utilized.

**Recommendations:**

These variable declarations in the function signature can be safely omitted as they do not impact the intended functionality of the code.

**Alleviation:**

Some of the corresponding functions of the Exhibit’s lines have been remediated, however most remain unchanged. Namely:

- CatalystMinter.sol - “tokenId” of “changeCatalyst”: L88 & L91
- CatalystMinter.sol - “Ids” of “mintMultiple”: L149 & L152
- CatalystRegistry.sol - “exists” & “catalystId” of “getCatalyst”: L12, L15, L19 & L21
- CatalystRegistry.sol - “emptySockets”, “index” & “seed” of “_getSocketData”: L88 - L90, L96, L101 & L103
- CatalystRegistry.sol - “values” of “getValues”: L75 & L76
- ERC20Group.sol - “supply” of “supplyOf”: L107, L109
- ERC20Group.sol - “balance” of “balanceOf”: L116, L118
- ERC20Group.sol - “balances” of “balanceOfBatch”: L125, L131
- ERC20Group.sol - “isOperator” of “isApprovedForAll”: L235, L236
- ERC20SubToken.sol - “success” of “transfer”: L53, L55
- ERC20SubToken.sol - “success” of “transferFrom”: L62, L72
- ERC20SubToken.sol - “success” of “approve”: L75, L77
- ERC20SubToken.sol - “success” of “approveFor”: L84, L87
- ERC20SubToken.sol - “remaining” of “allowance”: L111, L112

The Sandbox team has stated that these named variables act as a form of documentation and specification of the codebase and as such, it is normal to not adjust them based on our recommendation.

## Exhibit 5

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Redundant Value Assignment | Ineffectual Code | Informational | CatalystMinter.sol: L262 |

**[INFORMATIONAL] Description:**

When a variable is declared in Solidity it possesses a default value, usually equal to zero. As such, assignments of this default value to declared variables are unnecessary.

**Recommendations:**

The value assignment of “0” in the specified line can be safely removed.

**Alleviation:**

The value assignment was properly removed.

## Exhibit 6

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Duplicate Code Segments | Ineffectual Code | Informational | CatalystMinter.sol: L288 - L291 & L296 - L299 |

**[INFORMATIONAL] Description:**

The aforementioned code segments are exactly the same leading to unnecessary bytecode duplication.

**Recommendations:**

These segments should instead be replaced by a single “internal” function. Additionally, its necessity is questionable as it creates an incremental-value array to pass on to the function which may ultimately be unnecessary.

**Alleviation:**

The Sandbox team considered this Exhibit and chose to leave the code as is to avoid significant code changes in the codebase at its current state.

## Exhibit 7

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Visibility Specifiers Missing | Access Control | Informational | CatalystMinter.sol: L394, L395, L396, L397, L398, L399, L401, L402, L403, L404, L406<br>CatalystRegistry.sol: L128, L130<br>ERC20Group.sol: L366, L367 |

**[INFORMATIONAL] Description:**

Visibility specifiers are missing in the aforementioned lines that contain contract-level variable declarations.

**Recommendations:**

Visibility specifiers should be explicitly listed in Solidity code to avoid confusion of the intended purpose of a variable. Consider providing a visibility specifier for the variables included in the above location list.

**Alleviation:**

All the variables within the list were provided visibility specifiers by the Sandbox team.

## Exhibit 8

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Misleading Function Name | Coding Style | Minor | CatalystMinter.sol: L280 - L286 |

**[MINOR] Description:**

The function "_checkGemsQuantities" is meant to check the "gemQuantities" provided as arguments to the function, however it also alters them by subtracting 1 on each quantity before returning them.

**Recommendations:**

This type of behavior should either be properly documented directly on the codebase or lead to a change in the name of the function as further development of the project may misuse the function leading to potential vulnerabilities arising.

**Alleviation:**

The Sandbox team stated that this is the intended behavior of the corresponding code segment and that no documentation is necessary at this point in time as the team is fully aware of its intended purpose.

## Exhibit 9

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Magic Number Utilization | Coding Style | Informational | CatalystMinter.sol: L196 |

**[INFORMATIONAL] Description:**

The specified line contains the magic number “0xFFfFfFffFFffffFFfFFfFFFFFffFFFffffFfFFFfF” which is meant to represent the address that triggers a “burn” instead of a “transfer” transaction.

**Recommendations:**

As the number signifies a change of behavior for the function, we advise that it be stored in a “constant” variable within the contract. “constant” variables do not actually take up space on the contract and are replaced by the Solidity compiler at compile time, thus providing no gas overhead.

**Alleviation:**

The “constant” variable “BURN_ADDRESS” was properly declared to reflect the change of behavior in the function.

## Exhibit 10

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Unsafe Math Addition | Mathematical | Minor | CatalystMinter.sol: L274 |

**[MINOR] Description:**

A safe mathematics library is utilized throughout the codebase called “SafeMathWithRequire” except for the specified line whereby an unsafe addition is done with the result of a safe multiplication.

**Recommendations:**

As this action can overflow, we advise that this is replaced with a proper safe addition.

**Alleviation:**

The line in question was properly adapted to utilize the safe addition method of the “SafeMathWithRequire” library, nullifying this Exhibit.

## Exhibit 11

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Incorrect “SafeMathWithRequire” Implementation | Ineffectual Code | Minor | SafeMathWithRequire.sol: L29 |

**[MINOR] Description:**

This library is meant to replicate the exact statements of the “SafeMath” library albeit with “require” statements rather than “assert” statements. The re-implementation is correct barring for the “div” implementation which does not “require” that “b” is different from zero.

**Recommendations:**

While Solidity will indeed throw when dividing by zero as the comments indicate, it will not cause a “require” call but rather a low-level fail similar to “assert”. We advise that a “require” call is properly introduced in this function.

**Alleviation:**

A “require” call was properly introduced by the Sandbox team that conducts a “not-equal” (!=) comparison against zero.

## Exhibit 12

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Group Assignments to Struct | Unoptimized Code | Informational | CatalystRegistry.sol: L34 - L37 |

**[INFORMATIONAL] Description:**

The “storage” assignments of the aforementioned lines update all the fields of the “CatalystStored” struct at the specific index. Individually, each assignment needs to conduct specialized padding operations that increase their gas cost and conducts a full 256-bit update on storage.

**Recommendations:**

Instead, a single struct could be instantiated and assigned directly to omit the padding operations and store everything in a single storage assignment.

**Alleviation:**

The assignments were properly grouped to a single struct instantiation and assignment.

## Exhibit 13

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Invalid Bit-Wise Constants | Ineffectual Code | Minor | CatalystRegistry.sol: L81, L82 |

**[MINOR] Description:**

Within the smart contracts, two bit-wise operands that are contradictory exist named “IS_NFT” and “NOT_IS_NFT”.

**Recommendations:**

Judging by their utilization and variable names, when “IS_NFT” is true “NOT_IS_NFT” should be false and vice versa, however this is not the case for all values such as “0x0000000000000000000000000000000000000000A0000000000000000000000” due to the usage of “7” and “8” as the discerning factor whilst the representation is alphanumeric. We advise that the bit-wise operators are evaluated and ensured to fulfill their intended purpose.

**Alleviation:**

The Sandbox team has responded by stating that these bit-wise operands function as intended and their name is meant to guide the user with regards to what they are meant to represent rather than actually being opposite of each other.

## Exhibit 14

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Invalid “require” Message | Ineffectual Code | Informational | ERC20Group.sol: L50, L67 |

**[INFORMATIONAL] Description:**

The “require” call of the specified line checks whether the “msg.sender” is a minter, however the error message in case the check fails is “NOT_AUTHORIZED_ADMIN”.

**Recommendations:**

This error message should be altered to maintain that the user is not authorized as a minter rather than as an administrator, as the error “NOT_AUTHORIZED_ADMIN” is used under a different premise throughout the codebase.

**Alleviation:**

The error message was properly updated to reflect the correct type of access control being applied on both L50 and L67.

## Exhibit 15

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Inconsistent Implementation | Coding Style | Informational | ERC20Group.sol: L72 - L103, L168 - L210 |

**[INFORMATIONAL] Description:**

The functions “_batchMint” and “batchTransferFrom” are similar, however they utilize a different variable to conduct the logic and represent the maximum of “uint256” in two different ways (“2**256 - 1” and “0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”).

**Recommendations:**

We advise that these functions are refactored to utilize a single “internal” function that carries out the transfer loop as the code segments are identical. Additionally, we advise that the maximum of “uint256” throughout the codebase is represented via “~uint256(0)” to ensure consistency in the representation. “2**256 - 1” should be avoided as it may compile but would be unable to be conducted within Solidity due to overflow bounds.

**Alleviation:**

Upon further examination of both functions by the Sandbox team, a logical fault was identified that was the result of inconsistent implementation between the functions as they utilized different variables to conduct the conditional logic within the loop and outside of it.

This was properly amended by the Sandbox team and they decided to not proceed with grouping the logic in a single “internal” function per this Exhibit as including branching logic

within the “internal” function to satisfy both the requirements of “_batchMint” and “batchTransferFrom” would increase the gas cost substantially.

## Exhibit 16

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Unconventional “_name” and “_symbol” Storage | Ineffectual Code | Informational | ERC20SubToken.sol: L153, L154 |

**[INFORMATIONAL] Description:**

The contract at hand stores the “_name” and “_symbol” of an ERC-20 contract in raw “bytes32” format rather than a “string” format directly and consequently converts these raw byte formats to a “string” format in the corresponding getter functions of the variables.

**Recommendations:**

As the variables are not utilized elsewhere under their “bytes32” format, we advise that they are stored under their “string” representation directly.

**Alleviation:**

The variables were changed to the “string” format per our recommendation.

.

## Exhibit 17

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Inconsistent Library Utilization | Ineffectual Code | Informational | ERC20SubToken.sol: L130 |

**[INFORMATIONAL] Description:**

The “CatalystMinter.sol” contract is utilizing the “SafeMathWithRequire” library whereas the “ERC20SubToken.sol” contract is utilizing the “SafeMath” library directly.

**Recommendations:**

We advise that the same library is used across implementations, “SafeMathWithRequire”, to ensure that the debugging process of the system is uniform.

**Alleviation:**

The “SafeMath” library was properly replaced with its “WithRequire” counterpart in the “ERC20SubToken.sol” contract.

## Exhibit 18

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Value Literal “pure” Function Implementation | Ineffectual Code | Informational | ERC20SubToken.sol: L49 - L51 |

**[INFORMATIONAL] Description:**

The “decimals” function of the above lines returns the value literal of “0” casted to a “uint8”.

**Recommendations:**

As the function contains no other statements, we advise it be changed to a “public” and “constant” contract variable for which a getter is automatically generated.

**Alleviation:**

The Sandbox team decided to stick with the current representation as our recommended statements and the current statements of the codebase are identical in the generated bytecode.

## Exhibit 19

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Nested Duplicate Code Segments | Ineffectual Code | Informational | ERC20GroupCatalyst.sol: L11 - L25, L27 - L36 |

**[INFORMATIONAL] Description:**

The function “addCatalysts” internally replicates the statements of “addCatalyst” in L13 & L14.

**Recommendations:**

We advise that the functions are adapted to allow the utilization of an internal “_addCatalyst” function that does not possess the administrative access control and conducts the value overriding check.

**Alleviation:**

The benefit of grouping the statements into a single “internal” function was deemed negligible by the Sandbox team and as such no change was made in relation to this Exhibit.

## Exhibit 20

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Inefficient Order of “require” Calls | Optimization | Informational | CatalystMinter.sol: L273, L276 |

**[INFORMATIONAL] Description:**

The aforementioned “require” calls rely on data that is accessible earlier than their invocations.

**Recommendations:**

The aforementioned “require” calls can be directly put under the destructuring of the “_getMintData” call on L271 to ensure no superfluous statements are executed.

**Alleviation:**

The “require” calls were properly relocated to optimize the order of execution of the conditional statements.

## Exhibit 21

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Inefficient Memory-to-Memory Copy | Optimization | Informational | CatalystMinter.sol: L316, L317 |

**[INFORMATIONAL] Description:**

The declaration on L316 assigns a struct of the in-memory array of structs to an in-memory variable that is subsequently accessed by the line below it.

**Recommendations:**

The in-memory declaration is redundant as both variables are stored in-memory and as such, the in-memory array of structs can directly be accessed in the function invocation by providing the index on the accessor.

**Alleviation:**

Our recommendation was followed whereby the in-memory declaration of the struct was omitted from the loop's body.

## Exhibit 22

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Incorrect Function Name | Coding Style | Informational | CatalystDataBase.sol: L10<br>ERC20GroupCatalyst.sol: L20 |

**[INFORMATIONAL] Description:**

The function of “CatalystDataBase” is called “_setMindData” whereby it interacts with minting data.

**Recommendations:**

The function should be renamed to “_setMintData” as its arguments imply.

**Alleviation:**

The function was properly renamed to illustrate its intended purpose.

## Exhibit 23

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Misconception of Randomness | Ineffectual Code | Medium | CatalystDataBase.sol: L43 - L50, L78 |

**[MEDIUM] Description:**

The function “_computeValue” is meant to calculate a random value for assigning the value per gem ID. The result of its computation is stored in a variable named “randomValue”.

**Recommendations:**

Random values do not exist within a blockchain. All the variables the computation relies on are accessible by a miner before the submission of his block and, subsequently, the value of a gem ID is fully malleable by the miners of the Ethereum network. We advise estimating to what extent this affects the ecosystem of the platform and the renaming of the variable on L78 as it is completely misleading.

**Alleviation:**

The Sandbox team has stated that the rationale behind choosing these naming conventions is that the code is meant to replicate “randomness” to the greatest extent possible within a blockchain context, however the way it is utilized does not provide miners with any substantial benefit in affecting the values generated by the “_computeValue” function and as such it can be deemed safe.

## Exhibit 24

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Inexplicable Value Literal | Ineffectual Code | Minor | CatalystDataBase.sol: L82 |

**[MINOR] Description:**

The conditional check of the “getValues” function checks whether the value of a gem ID is zero and if so, computes it and assigns it to the “values” array. If it is non-zero, however, the value literal “25” is assigned instead of the actual “valuesPerGemIds” value.

**Recommendations:**

We advise that the purpose of assigning the literal “25” is explained and if none, the actual value of the “valuesPerGemIds” array is provided.

**Alleviation:**

The value literal “25” was properly documented in the codebase, thus nullifying this Exhibit.

## Exhibit 25

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Visibility Specifiers Missing | Coding Style | Informational | CatalystDataBase.sol: L113, L114 ERC20SubToken.sol: L146, L147 |

**[INFORMATIONAL] Description:**

The visibility specifiers of L113 and L114 & L146 and L147 are missing on CatalystDataBase and ERC20SubToken respectively.

**Recommendations:**

We advise that visibility specifiers are properly defined for these contract variables

**Alleviation:**

Visibility specifiers were added by the Sandbox team accordingly.

## Exhibit 26

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Potentially Ignored Struct Member | Ineffectual Code | Informational | CatalystDataBase.sol: L110 |

**[INFORMATIONAL] Description:**

The struct “MintData” contains a “uint16 maxGems” variable that is never read from and only assigned as a side-effect of “_setMintData”.

**Recommendations:**

Its purpose should be evaluated and if redundant, the codebase of the contract should be adjusted to reflect its omission accordingly.

**Alleviation:**

The Sandbox team responded by stating that the variable is indeed utilized by “CatalystMinter” and as such this Exhibit is nullified.

## Exhibit 27

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Inefficient Struct Member Assignment | Optimization | Informational | CatalystDataBase.sol: L26 - L29 |

**[INFORMATIONAL] Description:**

The statements between L26 and L29 can be grouped to a single instantiation and assignment as in its current state it conducts a full word update operation on each line.

**Recommendations:**

Included in the description above.

**Alleviation:**

The Sandbox team responded by stating that the function the statements are included in is seldomly invoked by the administrator and as such, they decided to retain the code as is.

## Exhibit 28

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Optimization of ID Logic | Optimization | Informational | CatalystDataBase.sol: L68 - L84 |

**[INFORMATIONAL] Description:**

The function “getValues” internally finds the maximum gem ID possible, instantiates an empty in-memory array with a length equal to that ID plus one and proceeds to utilize the array as a "mapping" of a gem's ID with its computed value.

**Recommendations:**

The code block can be optimized to instead not need to find the maximum gem ID nor instantiate an in-memory array of a substantial size in case the maximum gem ID is large enough.

To achieve this, a “private mapping(uint256 => mapping(uint256 => uint256))” can be utilized. The first "key" of the mapping could be an incremental nonce that tracks the number of invocations of the function.

The lookup operation could then be stored to an in-memory variable i.e. “mapping(uint256 => uint256) storage valuesPerGemIds” which would subsequently be accessed in the for loop of L75 to L84.

This would greatly optimize the gas cost of the function as the maximum gem ID loop would be redundant and no large array would be instantiated on each invocation.

**Alleviation:**

The Sandbox team responded by stating that the function is meant to be externally called only and as such, gas efficiency is of no importance for this particular function.

## Exhibit 29

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Value Overriddence Logic | Logical | Informational | ERC20GroupCatalyst.sol: L24 - L26 |

**[INFORMATIONAL] Description:**

The function “addCatalysts” accepts a “valueOverrides” array that signifies whether the values of a particular catalyst should be overridden. However, the logical check of L21 to L23 would make it impossible to override the value of the f.e. third catalyst without overriding all preceding values.

**Recommendations:**

An additional check should be imposed that ensures the value of “valueOverrides” isn't "empty". "empty" here could either mean 0, if the value is never expected to be that, or the maximum of “uint256”. In general, a reserved value should be used to designate “valueOverrides” that should be ignored.

**Alleviation:**

Comments that detail the above functionality were added to the function to properly document its behavior and the code was left as is to avoid major code changes prior to release.

## Exhibit 30

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Literal to Constant | Logical | Informational | ERC20Group.sol: L77, L82, L99, L294, L298, L317 |

**[INFORMATIONAL] Description:**

It is generally a widespread coding practice to store the maximum of an “uint256” to a “constant” contract variable if it is meant to be utilized across multiple segments of the code.

**Recommendations:**

We advise that the said value is stored to a “constant” contract value that is instead utilized across the codebase.

**Alleviation:**

The “constant” variable “MAX_UINT256” was declared by the Sandbox team.

## Exhibit 31

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Redundant Return | Ineffectual Code | Informational | ERC20Group.sol: L134 |

**[INFORMATIONAL] Description:**

The return statement of L134 can be safely omitted as the named return variable is utilized within.

**Recommendations:**

Included above.

**Alleviation:**

The return statement was properly omitted by the Sandbox team.

## Exhibit 32

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Utilization of Utility Functions | Coding Style | Informational | ERC20Group.sol: L150 - L157, L178 - L181, L223, L257 and L274 |

**[INFORMATIONAL] Description:**

The functions “isApprovedForAll”, “isAuthorizedToTransfer” and “isAuthorizedToApprove” are functions that could be used within the require statements of L150 to L157, L178 to L181, L223, L257 and L274. Additionally, “isAuthorizedToTransfer” internally can call “isAuthorizedToApprove”.

**Recommendations:**

Included above.

**Alleviation:**

The Sandbox team decided to avoid major code changes that would affect the generated bytecode of the contract at this point in time and thus did not apply this Exhibit.

## Exhibit 33

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Mapping Lookup Optimization | Optimization | Informational | ERC20Group.sol: L80 - L104, L186 - L211, L297 - L322 |

**[INFORMATIONAL] Description:**

The mapping lookups conducted in “_batchMint”, “batchTransferFrom” and “_batchBurnFrom” can be stored to an in-memory storage variable to optimize the gas cost of the functions significantly.

**Recommendations:**

In detail:

- _batchMint: The lookup “_packedTokenBalance[to]” can be stored to an in-memory variable
- batchTransferFrom: The lookups “_packedTokenBalance[to]” and “_packedTokenBalance[from]” can be stored to in-memory variables
- _batchBurnFrom: The lookup “_packedTokenBalance[from]” can be stored to an in-memory variable

The same optimization applies for all mapping lookups that occur twice, such as in “singleTransferFrom”, however the performance boost is crucial in the batch functions.

**Alleviation:**

The Sandbox team applied our recommendation in full on all functions, increasing the legibility of the codebase as well as its gas efficiency.

## Exhibit 34

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Redundant Safe Math | Ineffectual Code | Informational | ERC20SubToken.sol: L67, L68 |

**[INFORMATIONAL] Description:**

The statements of L67 and L68 conduct the same conditional checks whereby “allowance” is ensured to be greater than or equal to amount internally within the sub function of “SafeMathWithRequire”.

**Recommendations:**

We advise that either “SafeMathWithRequire” is updated to also accept calls with an error message, a convention followed by OpenZeppelin, or that the conditional of L67 is omitted. The former solution is more verbose and aids in the readability of the codebase.

The error message as input solution would also simplify require statements that appear throughout the codebase in other areas as well, such as L33 and L34 of CatalystRegistry.sol .

**Alleviation:**

The Sandbox team decided to avoid changing the “SafeMathWithRequire” library and instead chose to simply omit it from this contract and not utilize it to avoid duplicate comparisons.

## Exhibit 35

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Dead Code | Ineffectual Code | Informational | ERC20SubToken.sol: L123 - L127 |

**[INFORMATIONAL] Description:**

The function “_firstBytes32” between L123 and L127 is a leftover function of the previous iteration whereby the “_name” and “_symbol” variables were of type “bytes32”.

**Recommendations:**

We advise that the function is removed to reduce the bytecode of the compiled contract.

**Alleviation:**

The Sandbox team properly removed the function from the contract.

## Exhibit 36

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Unreachable Statement | Ineffectual Code | Informational | CatalystRegistry.sol: L106 - L111 |

**[INFORMATIONAL] Description:**

The documentation of L111 indicates that code should never reach this point.

**Recommendations:**

As such, we advise that an “assert” is utilized instead of an “if” clause between L106 and L110 to ensure that if “catalyst.set != 0” evaluates to “false”, it is asserted that “assetId & IS_NFT != 0” evaluates to “true”.

**Alleviation:**

The Sandbox team properly adjusted the code block to utilize an “assert” instead.