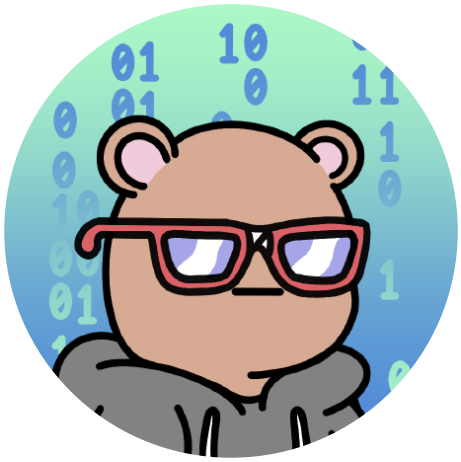

SanLeo Security

SNGLR Guilds Audit

March 15, 2025

Review commit hash

ef9ed65b1fc71d730eb4f279296cff44efad9f21

1 Disclaimer

Although the smart contracts analyzed in this report have been extensively scrutinized, no guarantees, express or implied, are made about the overall security or correctness of the software. It is important to remain vigilant with security long term; this potentially includes other audits, public auditing competitions, and bug bounty programs. SanLeo Security does not assume any liability.

2 Risk Classification

High	Medium	High	Critical
Medium	Low	Medium	High
Low	Low	Low	Medium
	Low	**Medium**	**High**

Severity: Likelihood vs Impact

A critical bug is defined as having a high likelihood of occurring and a high impact on the protocol, other bugs are a varying mix of likelihood and impact, for example an unlikely (low likelihood) bug can have a high impact, classifying it as a medium.

2.1 Impact

High - Causes substantial financial loss, significantly affects users, or breaks key invariants of the protocol.

Medium - Results in minor financial loss or impacts protocol functions in an unintended manner.

Low - Triggers unexpected behavior in the protocol, but has minimal consequences.

2.2 Likelihood

High - Exploitation is feasible under reasonable conditions and on-chain settings, and the cost of exploitation is low relative to financial impact.

Medium - Exploitation is possible but requires certain rarer conditions.

Low - Exploitation depends on highly unlikely circumstances or requires significant investment relative to financial impact.

3 Scope

```
1     evm/src/CoreUpgradeable.sol
```

4 Findings Count

Severity	Count
Critical	2
High	4
Medium	1
Low	7
Informational	9
Total	23

5 Summary of Findings

ID	Title	Status
[C-01]	runLottery is vulnerable to sniping attacks	Resolved
[C-02]	Can bypass totalSupply limit with reentrancy	Resolved
[H-01]	Can mint and run the lottery on the same block	Resolved
[H-02]	Missing token URI integration	Resolved
[H-03]	Missing ERC1155 name / symbol integration	Resolved
[H-04]	Missing royaltyInfo ERC165 check	Resolved
[M-01]	Minting grief vector	Resolved
[L-01]	runLottery may revert payout	Resolved
[L-02]	Guilds may be created with id 0	Resolved
[L-03]	Unrestricted mint data	Resolved
[L-04]	Artist royalty percentages are not validated	Resolved
[L-05]	totalBalances does not remain synchronized	Resolved
[L-06]	getTokenLotteryStatus returns incorrectly for nonexistent tokenId	Resolved
[L-07]	runLottery can fail in extraordinary circumstances	Resolved
[I-01]	View functions can be cleaner	Resolved
[I-02]	Missing ERC1155 initialization	Resolved
[I-03]	Typo in setMintSplit	Resolved
[I-04]	Contract not using Ownable2Step	Resolved
[I-05]	Internal functions not prefixed	Resolved
[I-06]	Uninitialized implementation contract	Acknowledged
[I-07]	Unused variables	Resolved
[I-08]	No method to change token URI	Resolved
[I-09]	Public functions should be marked as external	Resolved

6 Findings

6.1 Critical Findings

[C-01] runLottery is vulnerable to sniping attacks

```
656     uint256 winnerIndex =
657         uint256(keccak256(abi.encodePacked(
658             block.prevrandao, block.timestamp
659         ))) % (lotteryParticipants[_tokenId].length - 1);
660     tokens[_tokenId].winner = payable(lotteryParticipants[_tokenId][winnerIndex]);
```

Description
prevrandao and *block.timestamp* are not sufficient sources of randomness to draw a lottery. An attacker paying attention to the contract can wait for a block where the generated randomness favors them and call the *runLottery* function at will. The values of *prevrandao* and *block.timestamp* are known prior to each block. An attacker could optionally even bribe a block builder to guarantee that their transaction is included, or build a bribing contract to force a profit-focused builder to censor another transaction that would run the lottery, causing them to lose.

Recommendations
Use an established VRF (Verifiable Random Function) protocol, such as Chainlink, to generate secure randomness to draw lotteries. Alternatively, the *runLottery* function could be privileged to allow only the contract owner to draw the lottery, however, this introduces severe centralization and liveness risks.

[C-02] Can bypass totalSupply limit with reentrancy

```
462     function mint(
463         // ...
464     ) public payable whenNotPaused {
465         _validateMint(_to, _id, _value, _requestValidUntil, _total_price, _signature);
466
467         _mint(_to, _id, _value, _data);
468
469         // ...
470
471         tokens[_id].totalSupply += _value;
472     }
```

```
390     function _validateMint(
391         // ...
392     ) internal {
393         // ...
394         if (t.mintMaxSupply != 0 && (t.totalSupply + _value) > t.mintMaxSupply) {
395             revert MintOutOfSupply();
396         }
397     }
```

Description
The *ERC*1155 *_mint* function uses a safe transfer acceptance check, temporarily granting execution to the receiver of the token, if it is a contract. The receiving contract can then execute the *mint* function again, with a new signature that they have prepared. This new signature could even be for a different address/contract, as the reentering contract could transfer execution to another. This process can be continuously repeated until the attacker runs out of valid signatures, hits the gas limit for the block, or hits the call depth limit, at which point the total supply could be much higher than the allowed maximum limit.

Recommendations
Either add a reentrancy guard to the mint function, or use the Checks-Effects-Interactions (CEI) pattern to ensure that any reentrancy is effectively useless.

6.2 High Findings

[H-01] Can mint and run the lottery on the same block

```
390     function _validateMint(
391         // ...
392     ) internal {
393         // ...
394         if (block.timestamp > t.mintLaunchDate + (t.mintPeriodMinutes * 1 minutes)) {
395             revert MintClosed();
396         }
397         // ...
398     }
```

```
640     function runLottery(uint256 _tokenId) public returns (address) {
641         // ...
642         if (block.timestamp < tokens[_tokenId].mintLaunchDate + (tokens[_tokenId].
                mintPeriodMinutes * 1 minutes)) {
643             revert TooEarly();
644         }
645         // ...
646     }
```

Description

If the timestamp of the current block lands exactly on the ending timestamp of the mint window, it also becomes eligible for the lottery to be run in the same block. This may have a very significant chance of occurring on chains with high block speed. This vulnerability can be combined with [C-01] to potentially mint multiple times using different signatures, manipulating, testing, and reverting the outcome of the lottery until the attacker guarantees their winnings.

Recommendations

Change the operators $<$ and $>$ in the *runLottery* and *_validateMint* functions to use $<=$ and $>=$, ensuring that they revert correctly in this case.

[H-02] Missing token URI integration

Description

Although not a security vulnerability, this is being classified as a high-severity vulnerability for visibility, as this functionality is critical to an artwork-based contract.
When adding a token, the URI of the token itself is not set in the $ERC1155URIStorageUpgradeable$ contract; this means that marketplaces, indexers, and other NFT infrastructure will not be able to detect and render the artwork or metadata, as the *uri* function will simply return a blank string.

Recommendations

Set the URI of a token within the *addTokenId* method.

```
291     function addTokenId(
292         // ...
293     ) external virtual whenNotPaused returns (uint256 tokenId) {
294         // ...
295         counters.currentTokenId++;
296         tokens[counters.currentTokenId] = Token(
297             // ...
298         );
299         _setURI(counters.currentTokenId, _tokenUri);
300         // ...
301     }
```

[H-03] Missing ERC1155 name / symbol integration

```
149     function initialize(
150         string memory _name,
151         string memory _symbol,
152         // ...
153     ) public initializer {
154         // ...
155         MintSplitSettings memory ms;
156         settings = Settings(
157             _name,
158             _symbol,
159             // ...
160         );
161     }
```

Description

Although not a security vulnerability, this is being classified as a high-severity vulnerability for visibility, as this functionality is critical to an artwork-based contract.
The contract is initialized with a name and symbol intended to be used by chain explorers and marketplaces to index the collection; however, it does not expose these values in a standardized way. This would mean that chain explorers could default the collection to a name such as "ERC1155". Although inscribing the name and symbol in the contract itself is not technically in accordance with the specifications laid out in ERC1155, it has become standard practice to expose them via *name*() and *symbol*() functions to allow this functionality.

Recommendations

```
1     function name() external returns (string memory) {
2         return settings.name;
3     }
4
5     function symbol() external returns (string memory) {
6         return settings.symbol;
7     }
```

Expose the name and symbol in a standardized method, reading them from the *settings* variable, and returning them.

[H-04] Missing royaltyInfo ERC165 check

Description

Although not a security vulnerability, this is being classified as a high-severity vulnerability for visibility, as this functionality is critical to an artwork-based contract.
According to the specifications of ERC2981, the contracts that implement $royaltyInfo$ for marketplaces must also implement the ERC165 $supportsInterface$ check for the corresponding magic value. Failure to do this would likely mean marketplaces would not detect and, therefore, not pay out (or offer the option to pay out) any royalties.

Recommendations

Implement an overriding method for $supportsInterface$ that returns the correct magic value, according to the documentation, while ensuring to call super-interfaces such as $ERC1155URIStorageUpgradeable$ to allow them to comply with their specifications.

```
1     bytes4(keccak256("royaltyInfo(uint256,uint256)")) == 0x2a55205a
```

6.3 Medium Findings

[M-01] Minting grief vector

```
508         tokens[_id].artist.transfer(settings.mintSplit.artist * msg.value / 100);
```

Description
If the artist receiving payment upon a mint is a contract, it has the ability to revert upon receiving payment. It could even conditionally revert in order to create a blacklist of disallowed minters, or adding extra validations or bribes required to mint the artwork. Once this artist has been initialized for the corresponding token, there is no way to change it, meaning an artist can permanently grief a token.

Recommendations
Upon an ether transfer failure, force transfer using WETH.

6.4 Low Findings

[L-01] runLottery may revert payout

```
661     tokens[_tokenId].winner.transfer(tokens[_tokenId].lotteryFund);
```

Description
If the winner address is a contract that reverts upon receiving the funds, the contract will lose their winnings, and a new winner may be selected the next block. This is possibly not an issue; however, care must be taken if the *runLottery* function is altered to use VRF, as VRF cannot re-called if the transaction reverts. This would mean that the funds are permanently locked in the contract, so care must be taken to either force-transfer the assets using WETH or to allow for a redraw upon failure to transfer the funds.

Recommendations
Upon an ether transfer failure, force transfer using WETH, or allow an option to redraw if using VRF.

[L-02] Guilds may be created with id 0

```
308     function addGuild(uint256 _id) public onlyOwner {
309         if (guilds[_id].id != 0) {
310             revert GuildExist();
311         }
312
313         guilds[_id].id = _id;
314         counters.guildCount++;
315
316         emit GuildAdded(_id);
317     }
```

Description
The *addGuild* function does not validate that the inputted id is not equal to zero. If a guild is created with the id of zero, some strange effects take place.
In this case, the guild with id of 0 can be repeatedly recreated, incrementing the value of *guildCount* without actually creating a guild. It would also cause *getGuild* to revert when trying to retrieve guild details for id zero.

Recommendations
Verify in *addGuild* that the given *_id* is not zero.

[L-03] Unrestricted mint data

```
462     function mint(
463         // ...
464         bytes memory _data,
465         // ...
466     ) public payable whenNotPaused {
467         _validateMint(_to, _id, _value, _requestValidUntil, _total_price, _signature);
468 
469         _mint(_to, _id, _value, _data);
470         // ...
471     }
```

Description
The ERC1155 standard utilizes safe transfer acceptance checks for mints to ensure contracts do not erroneously receive tokens. The standard also allows the contract to send a value of *data* to instruct the receiving contract on how to act. The receiving contract does not trust this input implicitly, but allowing the user to input it directly could allow them to tamper with receiving contracts that may want to work with this extra data, if used, in the future.

Recommendations
Either include the data for the mint in the signature or hard code it to a default value.

[L-04] Artist royalty percentages are not validated

```
669     function setSecondaryMarketArtistSplitPercent(uint256 _percent) public onlyOwner {
670         settings.secondaryMarketArtistSplitPercent = _percent;
671     }
```

Description
The percentage set here by the owner could be 100 or greater, meaning that the sale price would be less than the royalties.

Recommendations
Add a check to ensure the royalties do not exceed a certain preset amount.

[L-05] totalBalances does not remain synchronized

```
498     totalBalances[msg.sender] += _value;
```

Description
The *totalBalances* mapping for a user is only updated when they mint an asset and is not decremented when they transfer any assets out; conversely, the *totalBalances* of a user receiving the asset will not be incremented.
This may not be a problem depending on the off-chain usage of this variable, as this variable does not have any direct impact on contract functionality.

Recommendations
Override the *_update* function in $ERC1155Upgradeable$ in order to update the value of *totalBalances* before the safe transfer acceptance check. Alternatively, deprecate the usage of this variable, and rely on the use of an off-chain indexer for this functionality.

[L-06] getTokenLotteryStatus returns incorrectly for nonexistent tokenId

```
619     function getTokenLotteryStatus(uint256 _tokenId) public view returns (string
            memory) {
620         Token memory token = tokens[_tokenId];
621         if (block.timestamp < token.mintLaunchDate + (token.mintPeriodMinutes * 1
                minutes)) {
622             return "WAIT_TO_MINT_END";
623         }
624 
625         if (token.lotteryFund == 0) {
626             return "NO_FUND";
627         }
628 
629         // ...
630     }
```

Description

If a corresponding *token* does not exist for the given *_tokenId*, the *getTokenLotteryStatus* function will return "NO_FUND" instead of "UNKNOWN" or reverting.

Recommendations

Add a check for if $mintLaunchDate == 0$ in *getTokenLotteryStatus*.

[L-07] runLottery can fail in extraordinary circumstances

```
640     function runLottery(uint256 _tokenId) public returns (address) {
641         // ...
```

```
653         if (tokens[_tokenId].totalSupply < settings.lottery.graduationLimit) {
654             tokens[_tokenId].winner = settings.wallets.lotteryFallback;
655         } else {
656             uint256 winnerIndex = uint256(keccak256(abi.encodePacked(block.prevrandao,
                    block.timestamp)))
657                 % (lotteryParticipants[_tokenId].length - 1);
658             tokens[_tokenId].winner = payable(lotteryParticipants[_tokenId][
                    winnerIndex]);
659         }
660 
661         // ...
662     }
```

Description

In the rare circumstance where $ticketsBeforeGraduations$ and $ticketsAfterGraduations$ are both 0, and *totalSupply* exceeds the *graduationLimit* there may be no tickets in the corresponding *lotteryParticipants* array. When this happens, upon calling *runLottery* the function will attempt to modulo by $length - 1$, when *length* is 0, this will revert. This will permanently lock the funds in the contract until a *lotteryParticipant* is added.

Recommendations

Add a check in line 653 to revert back to *lotteryFallback* if $lotteryParticipants[\_tokenId].length$ is 0.

6.5 Informational Findings

[I-01] View functions can be cleaner

```
676     function getSettings()
677         public
678         view
679         returns (
680             string memory,
681             string memory,
682     // ...
683     {
684         return (
685             settings.name,
686             settings.symbol,
687             settings.wallets.signer,
688     // ...
689             settings.lottery.ticketsBeforeGraduations,
690             settings.lottery.ticketsAfterGraduations,
691             settings.secondaryMarketArtistSplitPercent
692         );
693     }
```

Description
Its unnecessary, unclean, and difficult for contract upgrades and maintenance to destructure all variables in a view function.
This applies to other functions with the same behavior, namely *getToken*, *getGuild*, *getMintSplit*, and *getLotterySettings*.

Recommendations
Instead, the struct can be returned in its entirety.

```
676     function getSettings() public view returns (Settings memory _settings) {
677         _settings = settings;
678     }
```

This allows the reading client to destructure and read the variables using their variable names. If, for example, a contract upgrade was made, any code that relies on the current method may break or become vulnerable due to the ordering of variables being shifted around.

[I-02] Missing ERC1155 initialization

Description
Although initializing $ERC1155Upgradeable$ is technically unnecessary as the extension $ERC1155URIStorageUpgradeable$ is being used, leaving the slot in $ERC1155Upgradeable$ uninitialized is bad practice and could cause problems if contracts are upgraded in the future.

Recommendations

```
149     function initialize(
150         // ...
151     ) public initializer {
152         // ...
153         __ERC1155_init("");
154         // ...
155     }
```

Initialize ERC1155 in the initializer.

[I-03] Typo in setMintSplit

```
575     function setMintSpit(uint256 _artist, uint256 _snglr, uint256 _guilds, uint256
            _lottery) public onlyOwner
```

Description
The function, intended to be named *setMintSplit*, has been mistakenly written as *setMintSpit*.

Recommendations
Rename the function.

[I-04] Contract not using Ownable2Step

Description
The contract is currently inheriting from *OwnableUpgradeable*, making it particularly vulnerable to a mistake; if ownership is transferred to an incorrect wallet, it is permanent and irreversible.

Recommendations
Instead, use OpenZeppelins' *Ownable2StepUpgradeable* contract, this requires the receiving owner to accept the ownership transfer, greatly reducing the risk of a mistake.

[I-05] Internal functions not prefixed

```
446     function getTickets(uint256 _tokenId, uint256 _value) internal view returns (
            uint256) {
```

```
575     function getLotteryParticipants(uint256 id) internal view returns (address[]
            memory) {
```

Description
The internal functions should be prefixed with _ (underscore), according to solidity style guide.

Recommendations
Underscore the internal functions.

[I-06] Uninitialized implementation contract

Description
The implementation contract should initialize itself upon deployment; this is an additional layer of security typical of proxy contracts, which was adopted after a previous UUPS vulnerability.

Recommendations

```
1   constructor() {
2       _disableInitializers();
3   }
```

Add a constructor which disables the initializer in the implementation contract.

[I-07] Unused variables

Description
mintRoyalty and *mintRoyaltyCurrency* are set as parameters of the *Token* struct, and initialized within *addToken*, however, they are unused throughout the contract. Furthermore, it is strange that the variable *mintRoyaltyCurrency* is of type *uint*256, while an *address* would be more typical for a currency.

Recommendations
Remove the variables.

[I-08] No method to change token URI

Description
There is no method to change a given tokens URI, this is likely vital missing functionality, as external URIs are able to stop serving metadata at any point. Even IPFS URLs can either have no associated data or the network can lose the associated data. This is a common problem, and it would be unfortunate to lose the metadata and artwork of a token.

Recommendations
Add a method to edit the URI of a token.

[I-09] Public functions should be marked as external

Description
There are many functions in the contract that are unnecessarily marked as *public*. Functions that are *public* can be accessed both internally and externally from the contract, marking functions as *public* unnecessarily is not standard practice and can complicate the code.

Recommendations
Mark *public* functions that are not used internally as *external*.