NFT Marketplace Security Audit
Introduction

Audit Overview
We were tasked with performing an audit of the 10101 Art codebase and in particular their NFT collection creation mechanism and associated exchange and presale contracts.
Over the course of the audit, we identified a significant flaw in the way signature recovery behaves in the codebase as well as multiple minor misbehaviours across the codebase.
We advise the 10101 Art team to closely evaluate all minor-and-above findings identified in the report and promptly remediate them as well as consider all optimizational exhibits identified in the report.

Post-Audit Conclusion
The 10101 Art team iterated through all findings within the report and provided us with a revised commit hash to evaluate all exhibits on.
We evaluated all alleviations performed by 10101 Art and have identified that certain exhibits have not been adequately dealt with. We advise the 10101 Art team to revisit the following exhibits: WCF-01M, WCF-01C, ERT-01M, EOH-01M, ERC-02S, TPY-01M, PEL-05C, PEL-04C, PEL-03C, PEL-01M, MMR-01M, MMR-03M

Contracts Assessed

Files in Scope	Repository	Commit(s)
Address.sol (ASS)	smart-contracts	94c500b26d, 72bec452a7
Airdrop.sol (APO)	smart-contracts	94c500b26d, 72bec452a7
BytesLibrary.sol (BLY)	smart-contracts	94c500b26d, 72bec452a7
ERC721Factory.sol (ERC)	smart-contracts	94c500b26d, 72bec452a7
ExchangeState.sol (ESE)	smart-contracts	94c500b26d, 72bec452a7
ExchangeDomain.sol (EDN)	smart-contracts	94c500b26d, 72bec452a7

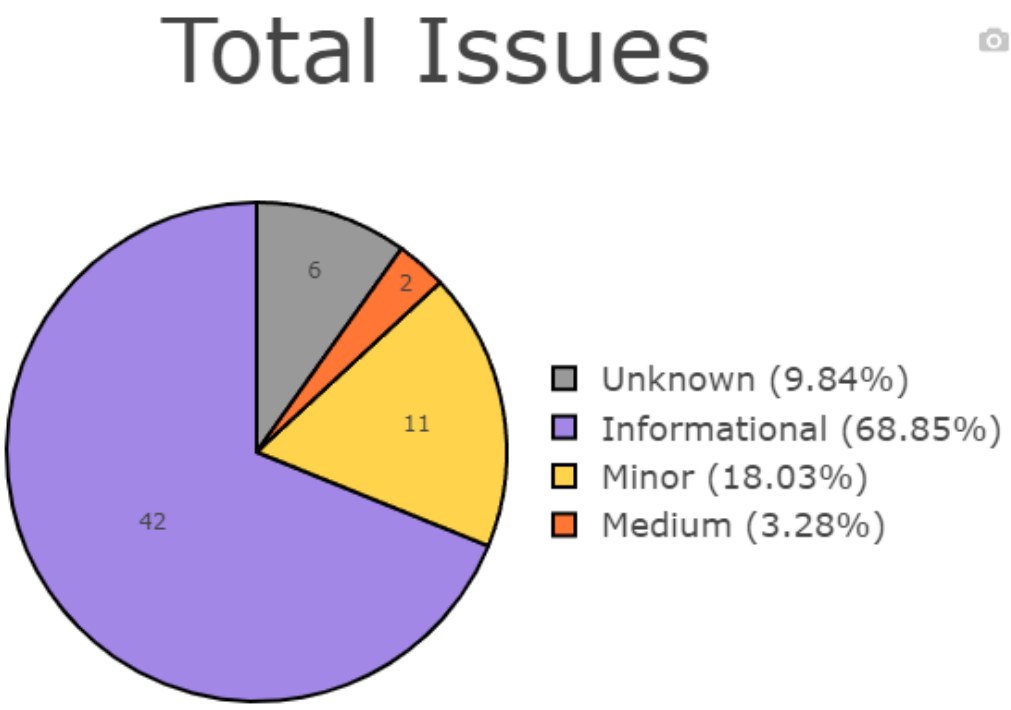
ERC721Collection.sol (ERN)	smart-contracts	94c500b26d, 72bec452a7
ERC20TransferProxy.sol (ERT)	smart-contracts	94c500b26d, 72bec452a7
ExchangeOrdersHolder.sol (EOH)	smart-contracts	94c500b26d, 72bec452a7
HasSecondarySaleFees.sol (HSS)	smart-contracts	94c500b26d, 72bec452a7
MarketMaker.sol (MMR)	smart-contracts	94c500b26d, 72bec452a7
OwnableExt.sol (OET)	smart-contracts	94c500b26d, 72bec452a7
Presale.sol (PEL)	smart-contracts	94c500b26d, 72bec452a7
SafeMath.sol (SMH)	smart-contracts	94c500b26d, 72bec452a7
TransferProxy.sol (TPY)	smart-contracts	94c500b26d, 72bec452a7
UintLibrary.sol (ULY)	smart-contracts	94c500b26d, 72bec452a7
WhitelistContractFilter.sol (WCF)	smart-contracts	94c500b26d, 72bec452a7

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	6	2	1	3

Informational	42	38	3	1
Minor	11	7	2	2
Medium	2	2	0	0
Major	0	0	0	0

During the audit, we filtered and validated a total of **11 findings utilizing static analysis** tools as well as identified a total of **50 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.



Compilation

The project utilizes `hardhat` as its development pipeline tool, containing an array of tests and scripts coded in JavaScript.
To compile the project, the `compile` command needs to be issued via the `npx` CLI tool to `hardhat`:

```
BASH
npx hardhat compile
```

The `hardhat` tool automatically selects Solidity version `0.8.9` based on the version specified within the `hardhat.config.js` file.
The project contains discrepancies with regards to the Solidity version used as the `pragma` statements of the contracts are open-ended (`^0.8.0`).

We advise them to be locked to 0.8.9 (=0.8.9), the same version utilized for our static analysis as well as optimizational review of the codebase.
During compilation with the hardhat pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

Static Analysis

The execution of our static analysis toolkit identified **149 potential issues** within the codebase of which **112 were ruled out to be false positives** or negligible findings.
The remaining **37 issues** were validated and grouped and formalized into the **11 exhibits** that follow:

ID	Severity	Addressed	Title
ERN-01S	Informational	Yes	Inexistent Event Emission
ERN-02S	Minor	Yes	Inexistent Sanitization of Input Address
ERC-01S	Informational	Yes	Inexistent Event Emissions
ERC-02S	Minor	Partial	Inexistent Sanitization of Input Addresses
MMR-01S	Informational	Yes	Inexistent Event Emissions
MMR-02S	Minor	Yes	Inexistent Sanitization of Input Addresses
OET-01S	Informational	Yes	Inexistent Event Emissions
PEL-01S	Medium	Yes	Improper Invocations of EIP-20 transfer / transferFrom
ULY-01S	Informational	Yes	Illegible Numeric Value Representation
WCF-01S	Informational	Yes	Inexistent Event Emissions
WCF-02S	Informational	Yes	Redundant Variable Assignment

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in 10101 Art's exchange and presale contract.
As the project at hand implements an EIP-721 and EIP-20 exchange, intricate care was put into ensuring that the **flow of funds & assets within the system conforms to the specifications and restrictions** laid forth within the protocol's specification.
We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **pinpointed an important vulnerability** within the system's signature recovery module which could have had **moderate ramifications** to its overall operation.
Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to a certain extent, however, we strongly recommend it to be expanded at certain complex points such as the various administrative functions that significantly centralize the project and appear unjustified.
A total of **50 findings** were identified over the course of the manual review of which **15 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.
The finding table below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
BLY-01M	Medium	Nullified	Insecure Elliptic Curve Signature Recovery Mechanism
ERT-01M	Unknown	No	Centralized Nature of Token Approvals
ERN-01M	Unknown	Yes	Arbitrary Burn Operation
ERN-02M	Minor	Yes	Potential Out of Gas Denial Attack
ERN-03M	Minor	Yes	Potentially Insufficient Override of ERC721A Functions
EOH-01M	Minor	Acknowledged	Inexistent Sanitization of Order
EOH-02M	Minor	Yes	Weak Existence Validation
MMR-01M	Unknown	Acknowledged	Arbitrary Approval Consumption
MMR-02M	Minor	Yes	Incorrect Payable Function Attribute
MMR-03M	Minor	No	Potentially Insecure Order Signature Validation

PEL-01M	Unknown	Partial	Inexistent Protection of State Transitions
PEL-02M	Minor	Yes	Inexistent Sanitization of Collection Creation
TPY-01M	Unknown	No	Centralized Nature of NFT Approvals
ULY-01M	Unknown	Yes	Outdated Strings Dependency Excerpt
WCF-01M	Minor	Partial	Incorrect Contract Removal Guards

Code Style

During the manual portion of the audit, we identified **35 optimizations** that can be applied to the codebase that will decrease the operational cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.
Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.
These optimizations are enumerated below:

ID	Severity	Addressed	Title
ASS-01C	Informational	Nullified	Outdated OpenZeppelin Dependency
APO-01C	Informational	Yes	Duplicate Application of Modifier
APO-02C	Informational	Yes	Event Practicality Enhancements
APO-03C	Informational	Yes	Inefficient Administrative Mint Workflow
APO-04C	Informational	Yes	Inefficient mapping Lookups
APO-05C	Informational	Yes	Loop Iterator Optimizations
BLY-01C	Informational	Nullified	Misleading Library Name

ERT-01C	Informational	Yes	Non-Standard Usage of Function Signature Literals
ERN-01C	Informational	Yes	Loop Iterator Optimization
ERN-02C	Informational	Yes	Redundant Conditional Structure
EDN-01C	Informational	Yes	Generic Typographic Mistake
EDN-02C	Informational	Nullified	Potential Data Structure Optimization
EOH-01C	Informational	Yes	Suboptimal Struct Declaration Style
ESE-01C	Informational	Yes	Discrepant Key Encoding Mechanism
HSS-01C	Informational	Yes	Event Practicality Enhancement
HSS-02C	Informational	Yes	Non-Standard Literal Definition of EIP-165 ID
MMR-01C	Informational	Yes	Non-Standard Definition of Unitary Maximum
MMR-02C	Informational	Yes	Non-Standard Literal Definition of EIP-165 ID
MMR-03C	Informational	Yes	Redundant Function Arguments
MMR-04C	Informational	Yes	Redundant Numeric Enum Comparison
MMR-05C	Informational	Yes	Redundant Payable Address Casts
MMR-06C	Informational	Nullified	Variable Mutability Specifier (Immutable)
OET-01C	Informational	Yes	Inconsistent State Transition Restrictions
PEL-01C	Informational	Yes	Duplicate Application of Modifier
PEL-02C	Informational	Yes	Generic Typographic Mistake

PEL-03C	Informational	Partial	Inefficient Data Pointers
PEL-04C	Informational	Partial	Inefficient mapping Lookups
PEL-05C	Informational	Acknowledged	Inexistent Specification of Override
PEL-06C	Informational	Yes	Loop Iterator Optimization
PEL-07C	Informational	Yes	Redundant Conditional Structure
SMH-01C	Informational	Nullified	Incorrect Usage of Dependency
WCF-01C	Informational	Partial	Inefficient mapping Lookups
WCF-02C	Informational	Yes	Loop Iterator Optimizations
WCF-03C	Informational	Yes	Redundant Duplication of Code
WCF-04C	Informational	Yes	Simplification of Ternary Operators

STATIC ANALYSIS

ERC721Collection Static Analysis Findings
ERN-01S: Inexistent Event Emission

Type	Severity	Location
Language Specific	Informational	ERC721Collection.sol:L53-L60

Description:

The linked function adjusts a sensitive contract variable yet does not emit an event for it.

Example:

contracts/ERC721Collection.sol

SOL Copy

```sol
53  function setWhitelistContractFilter(address _whitelistContractFilter)
54      external
55      onlyAdmin
56  {
57      whitelistContractFilter = WhitelistContractFilter(
58          _whitelistContractFilter
59      );
60  }
```

Recommendation:

We advise an `event` to be declared and correspondingly emitted to ensure off-chain processes can properly react to this system adjustment.

Alleviation:

A `ChangingWhitelistContractFilter` event has been introduced to the codebase and is correspondingly emitted in the **`ERC721Collection::setWhitelistContractFilter`** function, ensuring off-chain processes can adequately react to such an event.

ERN-02S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Minor	ERC721Collection.sol:L53-L60

Description:

The linked function accepts an `address` argument yet does not properly sanitize it.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

contracts/ERC721Collection.sol

SOL Copy

```sol
function setWhitelistContractFilter(address _whitelistContractFilter)
    external
    onlyAdmin
{
    whitelistContractFilter = WhitelistContractFilter(
        _whitelistContractFilter
    );
}
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that the `address` specified is non-zero.

Alleviation:

The input `address` of the `ERC721Collection::setWhitelistContractFilter` function is now adequately sanitized as non-zero, alleviating this exhibit.

Recommendation:
We advise some basic sanitization to be put in place by ensuring that the `address` specified is non-zero.

Alleviation:
The input `address` of the **`ERC721Collection::setWhitelistContractFilter`** function is now adequately sanitized as non-zero, alleviating this exhibit.

ERC721Factory Static Analysis Findings
ERC-01S: Inexistent Event Emissions

Type	Severity	Location
Language Specific	Informational	ERC721Factory.sol:L22-L30

Description:
The linked functions adjust sensitive contract variables yet do not emit an event for it.

Example:

contracts/ERC721Factory.sol

SOL Copy

```sol
constructor(
    address _presale,
    address _airdrop,
    address _whitelistContractFilter
) {
    presale = _presale;
    airdrop = _airdrop;
    whitelistContractFilter = _whitelistContractFilter;
}

/// @notice Function set new address contract Presale
/// @dev Only Admin
function setPresale(address _presale) external onlyAdmin {
    address oldPresale = presale;

    presale = _presale;

    emit ChangeAddressContract("Presale", oldPresale, presale);
}
```

Recommendation:
We advise an event to be declared and correspondingly emitted for each function to ensure off-chain processes can properly react to this system adjustment.

Alleviation:
Proper events have been introduced for all referenced variables, ensuring off-chain processes can adequately respond to their adjustment.

ERC-02S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Minor	ERC721Factory.sol:L22-L30, L34-L40, L44-L50, L54-L67

Description:
The linked function(s) accept address arguments yet do not properly sanitize them.

Impact:
The presence of zero-value addresses, especially in constructor implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

contracts/ERC721Factory.sol

SOL Copy

```sol
22  constructor(
23      address _presale,
24      address _airdrop,
25      address _whitelistContractFilter
26  ) {
27      presale = _presale;
28      airdrop = _airdrop;
29      whitelistContractFilter = _whitelistContractFilter;
30  }
```

Recommendation:
We advise some basic sanitization to be put in place by ensuring that each address specified is non-zero.

Alleviation:
While the input arguments of the **ERC721Factory::constructor** are now adequately sanitized, other referenced instances by the exhibit do not apply adequate sanitization rendering this exhibit partially alleviated.

MarketMaker Static Analysis Findings
MMR-01S: Inexistent Event Emissions

Type	Severity	Location
Language Specific	Informational	MarketMaker.sol:L75-L89, L93-L95, L99-L104, L107-L109, L112-L117

Description:
The linked functions adjust sensitive contract variables yet do not emit an event for it.

Impact:
93|95|94

Example:

contracts/NFT-Marketplace/MarketMaker.sol

SOL Copy

```sol
93  function setBeneficiary(address payable newBeneficiary) external onlyOwner {
94      beneficiary = newBeneficiary;
95  }
```

Recommendation:
We advise an event to be declared and correspondingly emitted for each function to ensure off-chain processes can properly react to this system adjustment.

Alleviation:
All referenced instances of variable adjustments are now accommodated by an event emission, ensuring off-chain processes can adequately react to their adjustment.

MMR-02S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Minor	MarketMaker.sol:L75-L89, L93-L95, L99-L104, L107-L109, L112-L

Description:
The linked function(s) accept `address` arguments yet do not properly sanitize them.

Impact:
The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

contracts/NFT-Marketplace/MarketMaker.sol

SOL Copy

```sol
constructor(
    TransferProxy _transferProxy,
    ERC20TransferProxy _erc20TransferProxy,
    ExchangeState _state,
    ExchangeOrdersHolder _ordersHolder,
    address payable _beneficiary,
    address _aliveUntilSigner
) {
    transferProxy = _transferProxy;
    erc20TransferProxy = _erc20TransferProxy;
    state = _state;
    ordersHolder = _ordersHolder;
    beneficiary = _beneficiary;
    aliveUntilSigner = _aliveUntilSigner;
}
```

Recommendation:
We advise some basic sanitization to be put in place by ensuring that each `address` specified is non-zero.

Alleviation:
All referenced instances of `address` variables are now adequately sanitized as non-zero, ensuring the contract cannot be misconfigured and alleviating this exhibit.

OwnableExt Static Analysis Findings
OET-01S: Inexistent Event Emissions

Type	Severity	Location
Language Specific	Informational	OwnableExt.sol:L32-L34, L39-L45

Description:
The linked functions adjust sensitive contract variables yet do not emit an event for it.

Example:

contracts/OwnableExt.sol

SOL Copy

```sol
32  function addAdmin(address _account) external onlyOwner {
33      admins[_account] = true;
34  }
```

Recommendation:
We advise an `event` to be declared and correspondingly emitted for each function to ensure off-chain processes can properly react to this system adjustment.

Alleviation:
All referenced instances of variable adjustments are now accommodated by an event emission, ensuring off-chain processes can adequately react to their adjustment.

Presale Static Analysis Findings
PEL-01S: Improper Invocations of EIP-20 `transfer` / `transferFrom`

Type	Severity	Location
Standard Conformity	Medium	Presale.sol:L127, L129, L200, L254

Description:
The linked statements do not properly validate the returned `bool` values of the **EIP-20** standard `transfer` & `transferFrom` functions. As the **standard dictates**, callers **must not** assume that `false` is never returned.

Impact:
If the code mandates that the returned `bool` is `true`, this will cause incompatibility with tokens such as USDT / Tether as no such `bool` is returned to be evaluated causing the check to fail at all times. On the other hand, if the token utilized can return a `false` value under certain conditions but the code does not validate it, the contract itself can be compromised as having received / sent funds that it never did.

Example:

contracts/Presale.sol

SOL Copy

```
127 erc20.transfer(ownerContract, amount);
```

Recommendation:
Since not all standardized tokens are **EIP-20** compliant (such as Tether / USDT), we advise a safe wrapper library to be utilized instead such as SafeERC20 by OpenZeppelin to opportunistically validate the returned bool only if it exists in each instance.

Alleviation:
All referenced instances of **EIP-20** transfer functions have been replaced by their safe-prefixed counterparts, alleviating this exhibit in full.

UintLibrary Static Analysis Findings

ULY-01S: Illegible Numeric Value Representation

Type	Severity	Location
Code Style	Informational	UintLibrary.sol:L36

Description:
The linked representation of a numeric literal is sub-optimally represented decreasing the legibility of the codebase.

Example:

contracts/libs/UintLibrary.sol

SOL Copy

```
36  return value.mul(bpValue).div(10000);
```

Recommendation:
To properly illustrate the value's purpose, we advise the following guidelines to be followed. For values meant to depict fractions with a base of 1e18, we advise fractions to be utilized directly (i.e. 1e17 becomes 0.1e18) as they are supported. For values meant to represent a percentage base, we advise each value to utilize the underscore (_) separator to discern the percentage decimal (i.e. 10000 becomes 100_00, 300 becomes 3_00 and so on). Finally, for large numeric values we simply advise the underscore character to be utilized again to represent them (i.e. 1000000 becomes 1_000_000).

Alleviation:
The underscore (_) character has been properly introduced to the referenced literal clearly denoting that it is expected to represent 100% with up to two decimal places of accuracy (100_00).

WhitelistContractFilter Static Analysis Findings
WCF-01S: Inexistent Event Emissions

Type	Severity	Location
Language Specific	Informational	WhitelistContractFilter.sol:L67-L74, L76-L82

Description:
The linked functions adjust sensitive contract variables yet do not emit an event for it.

Example:

contracts/WhitelistContractFilter.sol

SOL Copy

```sol
67  function addFilterPublic(address contractAccount) external onlyAdmin {
68      require(
69          isContract(contractAccount),
70          "The address you are trying to whitelist is not a contract!"
71      );
72
73      publicWhitelistContract[contractAccount] = true;
74  }
75
76  function removeFilterPublic(address contractAccount) external onlyAdmin {
77      require(
78          isContract(contractAccount),
79          "The address you are trying drop whitelist is not a contract!"
80      );
81      publicWhitelistContract[contractAccount] = false;
82  }
```

Recommendation:
We advise an event to be declared and correspondingly emitted for each function to ensure off-chain processes can properly react to this system adjustment.

Alleviation:
Proper events have been introduced for all referenced variables, ensuring off-chain processes can adequately respond to their adjustment.

WCF-02S: Redundant Variable Assignment

Type	Severity	Location

Gas Optimization	Informational	WhitelistContractFilter.sol:L20

Description:
The linked variable is assigned to redundantly to the default value of the relevant data type (i.e. `uint256` assigned to `0`, `address` assigned to `address(0)` etc.).

Example:

contracts/WhitelistContractFilter.sol

SOL Copy

```
20  bool public activeFilter = false;
```

Recommendation:
We advise the assignment to be safely omitted optimizing the codebase.

Alleviation:
The redundant variable assignment has been omitted, optimizing the contract's deployment cost.

BytesLibrary Manual Review Findings
BLY-01M: Insecure Elliptic Curve Signature Recovery Mechanism

Type	Severity	Location
Language Specific	Medium	BytesLibrary.sol:L12-L22

Description:
The `ecrecover` function is a low-level cryptographic function that should be utilized after appropriate sanitizations have been enforced on its arguments, namely on the `s` and `v` values. This is due to the inherent trait of the curve to be symmetrical on the x-axis and thus permitting signatures to be replayed with the same `x` value (`r`) but a different `y` value (`s`).

Impact:
Should the payload being verified by the signature rely on differentiation based on the `s` or `v` arguments, it will be possible to replay the signature for the same data validly and acquire authorization twice. Additionally, if the `aliveUntilSigner` member in `MarketMaker` is zero the `MarketMaker::validateAliveUntilSig` function can be bypassed by an arbitrary invalid signature being provided for the order payload.

Example:

contracts/libs/BytesLibrary.sol

SOL Copy

```sol
function recover(
    bytes32 message,
    uint8 v,
    bytes32 r,
    bytes32 s
) internal pure returns (address) {
    return
        ecrecover(
            keccak256(
                abi.encodePacked(
                    "\x19Ethereum Signed Message:\n32",
                    message
                )
            ),
            v,
            r,
            s
        );
}
```

Recommendation:
We advise them to be sanitized by ensuring that `v` is equal to either `27` or `28` (`v ∈ {27, 28}`) and to ensure that `s` is existent in the lower half order of the elliptic curve (`0 < s < secp256k1n ÷ 2 + 1`) by ensuring it is less than `0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A1`. A reference implementation of those checks can be observed in the **ECDSA** library of OpenZeppelin and the rationale behind those restrictions exists within **Appendix F of the Yellow Paper**. As a final point, the code should also evaluate that the result of `ecrecover` is not zero as that is the value returned for invalid signatures.

Alleviation:
The relevant file of the exhibit has been removed from the codebase rendering it no longer applicable.

ERC20TransferProxy Manual Review Findings
ERT-01M: Centralized Nature of Token Approvals

Type	Severity	Location
Centralization Concern	Unknown	ERC20TransferProxy.sol:L19, L39

Description:

The `erc20safeTransfer` / `erc20safeTransferFrom` functions permit an execution of `transfer` / `transferFrom` instructions for the contract's administrators which are controlled entirely by the contract's `owner`.

Example:

contracts/Proxies/ERC20TransferProxy.sol

SOL Copy

```sol
contract ERC20TransferProxy is OwnableExt {
    /// @notice Calls transferFrom for ERC20 and fails on error.
    /// @dev Can be called only by admin.
    /// @param addressToken Token ERC20
    /// @param from Account address from where to transfer
    /// @param to Account address where to transfer
    /// @param value Amount tokens ERC20
    function erc20safeTransferFrom(
        address addressToken,
        address from,
        address to,
        uint256 value
    ) external onlyAdmin {
        (bool success, bytes memory data) = addressToken.call(
            abi.encodeWithSelector(0x23b872dd, from, to, value)
        );

        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "TRANSFER_FROM_FAILED"
        );
    }

    /// @notice Calls transfer for ERC20 and fails on error.
    /// @dev Can be called only by admin.
    /// @param addressToken Token ERC20
    /// @param to Account address where to transfer
    /// @param value Amount tokens ERC20
    function erc20safeTransfer(
        address addressToken,
        address to,
        uint256 value
    ) external onlyAdmin {
```

Recommendation:

We advise the ownership structure of the contract to be revised and potentially made autonomous by eliminating ownership once the administrators necessary for the 10101 Art system to function have been defined.

Alleviation:
The ownable structure has been removed entirely from the ERC20TransferProxy contract, rendering it insecure as any approval to it can be arbitrarily consumed. We advise the ownership structure to be reverted. To note, we advised the ownership structure to be renounced once the administrators have been set; not to omit ownership entirely.

ERC721Collection Manual Review Findings
ERN-01M: Arbitrary Burn Operation

Type	Severity	Location
Centralization Concern	Unknown	ERC721Collection.sol:L79-L81

Description:
The code of ERC721Collection permits its administrators to arbitrarily burn token IDs from its users without validating any approval.
Example:

contracts/ERC721Collection.sol

SOL

Copy

```sol
76  /// @notice Collection token burn function
77  /// @dev Only Admin
78  /// @param tokenId Token Id collection
79  function burn(uint256 tokenId) external onlyAdmin {
80      _burn(tokenId);
81  }
```

Recommendation:
We advise the burn functionality to be omitted from the code as it appears to not be in use throughout the 10101 Art ecosystem. Alternatively, we advise an approval to be validated between the owner of the token ID and the caller of the function to ensure burn operations are authorized.
Alleviation:
The **ERC721Collection::burn** function has been omitted as advised.

ERN-02M: Potential Out of Gas Denial Attack

Type	Severity	Location
Language Specific	Minor	ERC721Collection.sol:L96-L108

Description:
The burnAll function is meant to iterate through all minted IDs and emit a Transfer event for each to signal that it has been burned. Given that a block has a limited gas limit, it may be impossible to invoke burnAll if many tokens have been minted.
Impact:

A user can presently detect whether a collection is to-be-burned and purchase a significant amount of tokens from the Presale contract to prohibit the burn operation from succeeding.

Example:

contracts/ERC721Collection.sol

SOL Copy

```sol
/// @notice The function of burning all tokens of the collection
/// @dev Only Admin
function burnAll() external onlyAdmin {
    isBurnt = true;
    maxSupply = 0;

    uint256 totalSupply = totalSupply();

    require(
        totalSupply != 0,
        "The operation is not possible because there is nothing to burn!"
    );

    unchecked {
        uint256 currentTokenId = totalSupply;

        do {
            currentTokenId = currentTokenId - 1;

            emit Transfer(
                ERC721A.ownerOf(currentTokenId),
                address(0),
                currentTokenId
            );
        } while (currentTokenId != 0);
    }
}
```

Recommendation:

We advise the code to instead set the isBurnt variable immediately and to consequently emit the Transfer events in sequence, permitting the user to pause the sequence and resume it at a secondary transaction. In turn, this will guarantee that regardless of the amount of token IDs minted the collection will be burnable and the events emittable albeit potentially in multiple transactions.

Alleviation:

The code of **ERC721Collection::burnAll** was updated to accept a new amount argument that denotes the number of NFTs that should be burned in the transaction. As the isBurnt variable is set immediately and is in use throughout the contract's transfer-related functions, we consider this exhibit adequately alleviated as the function will resume at the point it left off in the previous invocation as advised.

ERN-03M: Potentially Insufficient Override of ERC721A Functions

Type	Severity	Location

Standard Conformity	Minor	ERC721Collection.sol:L127-L131

Description:
The ERC721Collection contract is meant to apply transfer control checks to the target as well as about whether the collection has been burnt, however, this is inadequately applied.

Impact:
While the current implementation is adequate for ERC721A implementations of version 4 and up, it would not have properly behaved in versions 3 and below. As such, the _beforeTokenTransfers hook should be adequately overridden to ensure the code behaves consistently across ERC721A versions.

Example:

contracts/ERC721Collection.sol

SOL Copy

```sol
function transferFrom(
    address from,
    address to,
    uint256 tokenId
) public override checkIsBurnt {
    require(
        checkApproval(to),
        "TransferFrom to this address is prohibited by the whitelist filter!"
    );

    ERC721A.transferFrom(from, to, tokenId);
}
```

Recommendation:
We advise the code to also override the _beforeTokenTransfers implementation instead to ensure that all public-facing functions properly disallow transfer of assets when the collection is burnt as well as when the recipient is not sufficiently approved by the whitelist mechanism. As a final note, the code should also override the totalSupply and balanceOf functions of ERC721A to yield 0 if the collection has been burned as they will presently yield misleading values.

Alleviation:
The ERC721Collection::_beforeTokenTransfers function that has been overridden now properly applies approval checks to the recipient via the **ERC721Collection::checkApproval** function as advised, ensuring the correct checks are applied in all types of transfers performed with the **EIP-721** asset.

ExchangeOrdersHolder Manual Review Findings
EOH-01M: Inexistent Sanitization of Order

Type	Severity	Location

Input Sanitization	Minor	ExchangeOrdersHolder.sol:L24

Description:
The order that is being registered in the system remains unsanitized.
Impact:
An abnormal fee will cause the order to be unfulfillable due to unserviceable fees.
Example:

contracts/NFT-Marketplace/ExchangeOrdersHolder.sol

SOL Copy

```
21  /// @notice This function can be called to add the order to the contract, so i
22  ///         Can be called only by the order owner.
23  /// @param order - The order struct to add.
24  function add(ExchangeDomain.Order calldata order) external {
25      require(
26          msg.sender == order.key.owner,
27          "order could be added by owner only"
28      );
29      bytes32 key = prepareKey(order);
30      require(
31          orders[key].selling == 0 &&
32              orders[key].buying == 0 &&
33              orders[key].sellerFee == 0,
34          "Order is already existed. Try to change salt"
35      );
36      orders[key] = OrderParams(order.selling, order.buying, order.fee);
37  }
```

Recommendation:
We advise the order.fee to be mandated as at most equivalent to 100_00, the limit expected by the UintLibrary::bp implementation.
Alleviation:
The 10101 Art team has opted to not apply a remediation for this exhibit instead acknowledging it.

EOH-02M: Weak Existence Validation

	ity	ion
al Fault	linor	ngeOrdersHolder.sol:L42-L53

Description:
The exists function of ExchangeOrdersHolder is highly sensitive and is meant to be utilized by MarketMaker::exchange to validate that the owner of an order has authorized a sale.

Impact:
Although the possibility of a collision is negligible, the code does not validate who created an order which is a counter-intuitive approach to validating an order's presence.

Example:

contracts/NFT-Marketplace/ExchangeOrdersHolder.sol

SOL Copy

```sol
/// @notice This function checks if order was added to the orders holder contr
/// @param order - The order struct to check.
/// @return true if order is present in the contract's data.
function exists(ExchangeDomain.Order calldata order)
    external
    view
    returns (bool)
{
    bytes32 key = prepareKey(order);
    OrderParams memory params = orders[key];
    return
        params.buying == order.buying &&
        params.selling == order.selling &&
        params.sellerFee == order.fee;
}
```

Recommendation:
We advise the code of exists to also validate and store the order.owner to the params of a particular key, ensuring that even if a key collision is artificially crafted the owner will still be the authorizing party of the sale.

Alleviation:
The owner is now validated as being equivalent in both the params and order.key entry, alleviating this exhibit.

MarketMaker Manual Review Findings
MMR-01M: Arbitrary Approval Consumption

Type	Severity	Location
Centralization Concern	Unknown	MarketMaker.sol:L155

Description:
The exchange function permits an administrator to set arbitrary buyer members and thus consume arbitrary approvals of users when performing an exchange.

Impact:
Administrators are currently able to tap into the approvals of any party to the exchange, potentially compromising their assets.

Example:

contracts/NFT-Marketplace/MarketMaker.sol

SOL Copy

```sol
function exchange(
    Order calldata order,
    ECDSASig calldata sig,
    uint256 aliveUntil,
    ECDSASig calldata aliveUntilSig,
    uint256 amount,
    address buyer
) external payable {
    validateOrderSig(order, sig);
    validateAliveUntilSig(order, aliveUntil, aliveUntilSig);
    uint256 paying = order.buying.mul(amount).div(order.selling);
    verifyOpenAndModifyOrderState(order.key, order.selling, amount);

    FeeSide feeSide = getFeeSide(
        order.key.sellAsset.assetType,
        order.key.buyAsset.assetType
    );

    if (buyer == address(0x0)) {
        buyer = msg.sender;
    } else {
        require(
            admins[msg.sender],
            "Invalid buyer because the caller is not allowed to set its own bu
        );
    }
```

Recommendation:

We advise the code to prohibit such an action, only permitting the `buyer` to be the `msg.sender`. As an additional point, this change would allow the `buyer` argument to be omitted entirely as it would no longer be in use.

Alleviation:

The 10101 Art team has specified that this is part of their business requirements and that they wish to be able to provide an arbitrary `buyer` argument to the function to fulfil an exchange. The responsible party (`admin`) would be an authorized member of the 10101 Art team and thus is meant to act as a trustworthy entity in the 10101 Art system. As a result, we consider this exhibit acknowledged given that it represents a desirable business requirement by 10101 Art.

MMR-02M: Incorrect Payable Function Attribute

Type	Severity	Location
Language Specific	Minor	MarketMaker.sol:L140

Description:
The referenced function is set as `payable` yet does not make use of native funds either at rest or per transaction.

Impact:
It is currently possible for native funds to be permanently locked in the contract if they are sent alongside an `exchange` call which is an undesirable trait.

Example:

contracts/NFT-Marketplace/MarketMaker.sol

SOL Copy

```sol
133 function exchange(
134     Order calldata order,
135     ECDSASig calldata sig,
136     uint256 aliveUntil,
137     ECDSASig calldata aliveUntilSig,
138     uint256 amount,
139     address buyer
140 ) external payable {
```

Recommendation:
We advise the `payable` keyword to be safely omitted from the function's declaration.

Alleviation:
The incorrect `payable` attribute has been safely omitted as advised.

MMR-03M: Potentially Insecure Order Signature Validation

Type	Severity	Location
Logical Fault	Minor	MarketMaker.sol:L206-L207

Description:
The `exchange` system of `MarketMaker` validates signature per order creator and not per buyer, allowing an on-chain race-condition to occur whereby a different buyer can use the same signature that may have been privately provided by the `owner` to a single buyer.

Impact:
The system is presently prone to race conditions and can cause a user to not acquire the assets they hoped for via the `exchange` function.

Example:

contracts/NFT-Marketplace/MarketMaker.sol

SOL Copy

```sol
198 function validateOrderSig(Order memory order, ECDSASig memory sig)
199     internal
200     view
201 {
202     if (sig.v == 0 && sig.r == bytes32(0x0) && sig.s == bytes32(0x0)) {
203         require(ordersHolder.exists(order), "incorrect signature order");
204     } else {
205         require(
206             prepareMessage(order).recover(sig.v, sig.r, sig.s) ==
207                 order.key.owner,
208             "incorrect signature order"
209         );
210     }
211 }
```

Recommendation:

We advise the signature validation code to also ensure that the buyer is included in the prepareMessage of validateOrderSig, preventing other users from using the same signature to purchase a potentially private offer by an owner.

Alleviation:

The 10101 Art team has specified that they wish to retain the current behaviour in place as it contrasts their intended business requirements. We would like to denote that the finding relates to sales meant to be consumed by a single buyer, as in a private sale. In such a case, the signature validation mechanism should ensure that the buyer is also part of the validated payload.
In case the sale has an arbitrary recipient (scenario described by the 10101 Art team), the current signature validation mechanism can remain in place.

Presale Manual Review Findings

PEL-01M: Inexistent Protection of State Transitions

Type	Severity	Location
Centralization Concern	Unknown	Presale.sol:L104-L110, L116-L131

Description:

The burnAll and withdraw functions of the contract are meant to permit the administrators to perform sensitive state transitions, however, no checks are applied to ensure those transitions are correct.

Impact:

The contract does not presently contain any guarantees to its users and permits the administrators to extract user funds as well as destroy user assets at will.

Example:

contracts/Presale.sol

SOL

Copy

```sol
function burnAll(address collection) external onlyAdmin {
    ERC721Collection erc721 = ERC721Collection(collection);

    erc721.burnAll();

    emit BurningTokens(collection);
}

/// @notice The function of transferring ERC20 tokens from the contract to the
/// @dev Only Admin
/// @param erc20Address Address ERC20
/// @param amount The Number ER20 token
function withdraw(address erc20Address, uint256 amount) external onlyAdmin {
    IERC20 erc20 = IERC20(erc20Address);
    uint256 balanceContract = erc20.balanceOf(address(this));
    address ownerContract = owner();

    require(
        balanceContract != 0,
        "There is nothing on the balance of the contract now."
    );

    if (amount <= balanceContract) {
        erc20.transfer(ownerContract, amount);
    } else {
        erc20.transfer(ownerContract, balanceContract);
    }
}
```

Recommendation:

We advise the `burnAll` function to be invoke-able on a `collection` only if its sale is in progress, permitting users to withdraw their funds via `returnFunds` properly. As a next step, the `withdraw` function should be invoke-able per `collection` rather than per `erc20Address` and should only extract the funds that were raised during a collection's sale which need to be tracked. To prevent malicious behaviour, the `withdraw` function should not be invoke-able while a sale is in progress (disallowing the administrators from withdrawing funds and then burning the collection) or when a collection has been burned (disallowing the administrators from withdrawing funds meant to be refunded to users).

Alleviation:

While the state transitions of the **`Presale::burnAll`** and **`Presale::withdraw`** functions adequately sanitize the current state of a sale, they do not impose any limitation on the input `_amount` thus partially alleviating this exhibit.

PEL-02M: Inexistent Sanitization of Collection Creation

Type	Severity	Location
Input Sanitization	Minor	Presale.sol:L57-L63

Description:

The `addCollection` does not apply any form of sanitization in its input arguments, permitting incorrect presale configurations to be created for a collection.

Impact:

Misconfigured presales will fail to function properly and will cause misbehaviours in how funds are accepted by the contract.

Example:

contracts/Presale.sol

SOL

Copy

```sol
function addCollection(
    address collection,
    uint256 whitelistPrice,
    uint256 publicPrice,
    uint256 startWhitelistTimestamp,
    uint256 startPublicTimestamp,
    uint256 stopWhitelistTimestamp,
    uint256 stopTimestamp,
    address erc20Address
) external onlyAdmin checkRemoveCollection(collection) {
    Collection memory newCollection = Collection({
        whitelistPrice: whitelistPrice,
        publicPrice: publicPrice,
        startWhitelistTimestamp: startWhitelistTimestamp,
        startPublicTimestamp: startPublicTimestamp,
        stopWhitelistTimestamp: stopWhitelistTimestamp,
        stopTimestamp: stopTimestamp,
        erc20Address: erc20Address
    });

    collections[collection] = true;
    collectionInformations[collection] = newCollection;

    emit AddingCollection(collection);
}
```

Recommendation:

We advise the code to properly ensure that the `whitelistPrice` is lower than the `publicPrice`, the `startWhitelistTimestamp` is less than the `stopWhitelistTimestamp` which is less than the `startPublicTimestamp`

that in turn is less than the `stopTimestamp`. As a final check, the code should also validate that the `erc20Address` is non-zero.

Alleviation:

A `Presale::_beforeAddCollection` hook was introduced that applies all the recommended sanitizations with an exception to the `startWhitelistTimestamp` and `startPublicTimestamp` which should only be compared between them and not with `stopWhitelistTimestamp` per the business requirements of 10101 Art. As a result, we consider this exhibit fully alleviated.

TransferProxy Manual Review Findings

TPY-01M: Centralized Nature of NFT Approvals

Type	Severity	Location
Centralization Concern	Unknown	TransferProxy.sol:L21

Description:

The `erc721safeTransferFrom` function permits an execution of a `safeTransferFrom` instruction for the contract's administrators which are controlled entirely by the contract's `owner`.

Example:

contracts/Proxies/TransferProxy.sol

SOL

Copy

```sol
contract TransferProxy is OwnableExt {
    /// @notice Calls safeTransferFrom for ERC721.
    /// @dev Can be called only by admin.
    /// @param token Token ERC721
    /// @param from Account address from where to transfer
    /// @param to Account address where to transfer
    /// @param tokenId Token Id ERC721
    function erc721safeTransferFrom(
        IERC721 token,
        address from,
        address to,
        uint256 tokenId
    ) external onlyAdmin {
        token.safeTransferFrom(from, to, tokenId);
    }
```

Recommendation:

We advise the ownership structure of the contract to be revised and potentially made autonomous by eliminating ownership once the administrators necessary for the 10101 Art system to function have been defined.

Alleviation:

The ownable structure has been removed entirely from the TransferProxy contract, rendering it insecure as any approval to it can be arbitrarily consumed. We advise the ownership structure to be reverted. To note, we advised the ownership structure to be renounced once the administrators have been set; not to omit ownership entirely.

UintLibrary Manual Review Findings
ULY-01M: Outdated Strings Dependency Excerpt

Type	Severity	Location
Standard Conformity	Unknown	UintLibrary.sol:L9-L29

Description:
The referenced code represents an excerpt of the Strings library by OpenZeppelin, however, an outdated one is in use that may also malfunction as its arithmetic statements are meant to be executed unsafely.

Example:

contracts/libs/UintLibrary.sol

SOL Copy

```sol
function toString(uint256 value) internal pure returns (string memory) {
    // Inspired by OraclizeAPI's implementation - MIT licence
    // https://github.com/oraclize/ethereum-api/blob/b42146b063c7d6ee1358846c1

    if (value == 0) {
        return "0";
    }
    uint256 temp = value;
    uint256 digits;
    while (temp != 0) {
        digits++;
        temp /= 10;
    }
    bytes memory buffer = new bytes(digits);
    while (value != 0) {
        digits -= 1;
        buffer[digits] = bytes1(uint8(48 + uint256(value % 10)));
        value /= 10;
    }
    return string(buffer);
}
```

Recommendation:

We advise the latest version of `Strings` in the OpenZeppelin repository to be consulted and its code carried over to the `UintLibrary` to ensure that it behaves as expected.

Alleviation:

The **`UintLibrary::toString`** implementation has been updated to the latest one by OpenZeppelin, greatly optimizing its gas cost.

WhitelistContractFilter Manual Review Findings

WCF-01M: Incorrect Contract Removal Guards

Type	Severity	Location
Input Sanitization	Minor	WhitelistContractFilter.sol:L58, L78, L120, L158

Description:

The referenced `isContract` checks are applied when removing a contract from the whitelist, however, a contract can pass the `isContract` when being included and fail it when being excluded if it has been `selfdestruct`-ed for example.

Impact:

Presently, entries that may have ephemerally passed the `isContract` test will not be removable which is an undesirable trait, especially with contracts such as `create2` clones which can be redeployed.

Example:

contracts/WhitelistContractFilter.sol

SOL Copy

```sol
/// @notice Function add filter element for element private WhitelistContract
/// @dev Public function. Only Admin
/// @param element address element for WhitelistContractFilter
/// @param contractAccount address filter for element
function addFilterPrivate(address element, address contractAccount)
    public
    onlyAdmin
{
    require(
        isContract(contractAccount),
        "The address you are trying to whitelist is not a contract!"
    );

    privateWhitelistContract[element][contractAccount] = true;

    emit AddingApproveContractAccount(element, contractAccount);
}

/// @notice Function remove filter element for element private WhitelistContra
/// @dev Public function. Only Admin
/// @param element address element for WhitelistContract
/// @param contractAccount address filter for element
function removeFilterPrivate(address element, address contractAccount)
    public
    onlyAdmin
{
    require(
        isContract(contractAccount),
        "The address you are trying drop whitelist is not a contract!"
    );

    privateWhitelistContract[element][contractAccount] = false;

    emit RemovingApproveContractAccount(element, contractAccount);
}
```

Recommendation:

We advise the code to not apply an `isContract` check when removing an element from the whitelists.

Alleviation:

The WhitelistContractFilter::isContract check was omitted from both inclusions and removals of filters contrary to what we advised. We advise the contract validation to be re-instated for the WhitelistContractFilter::addFilterPrivate / WhitelistContractFilter::addFilterPublic functions as they should still validate the included filter is a contract.

Address Code Style Findings

ASS-01C: Outdated OpenZeppelin Dependency

Type	Severity	Location
Gas Optimization	Informational	Address.sol:L7

Description:

The Address dependency in use by the codebase represents an outdated version of the Address contract by OpenZeppelin.

Example:

contracts/libs/Address.sol

SOL Copy

```sol
84  function functionCall(address target, bytes memory data)
85      internal
86      returns (bytes memory)
87  {
88      return functionCall(target, data, "Address: low-level call failed");
89  }
```

Recommendation:

We advise the latest version to be utilized as it is more optimal than the one currently in use by the codebase.

Alleviation:

The relevant file of the exhibit has been removed from the codebase rendering it no longer applicable.

Airdrop Code Style Findings

APO-01C: Duplicate Application of Modifier

Type	Severity	Location
Gas Optimization	Informational	Airdrop.sol:L154

Description:

The checkAddCollection modifier is applied by the isWhitelist function as well as the getTokens function it is invoked in.

Example:

contracts/Airdrop.sol

SOL Copy

```sol
/// @notice Function for issuing NFT collection tokens for free
/// @dev Only Whitelist for users
/// @param collection Address NFT collection
/// @param proofs A set of proofs to confirm that an account is whitelisted
/// @param tokenAmount Number token for airdrop
/// @param maxAmount Max Number token for airdrop. For check amount
function getTokens(
    address collection,
    bytes32[] calldata proofs,
    uint256 tokenAmount,
    uint256 maxAmount
) external virtual checkAddCollection(collection) {
    bytes32 accountHash = keccak256(
        abi.encodePacked(msg.sender, maxAmount)
    );

    require(
        isWhitelist(collection, proofs, accountHash),
        "Account is not whitelisted."
    );

    _getTokens(msg.sender, collection, tokenAmount, maxAmount);
}

/// @notice Function for issuing NFT collection tokens for free
/// @dev Only Admin
/// @param collection Address NFT collection
/// @param airdropAccounts Account Dataset
function getTokensAdmin(
    address collection,
    AirdropToken[] calldata airdropAccounts
) external onlyAdmin checkAddCollection(collection) {
    for (uint256 i = 0; i < airdropAccounts.length; i++) {
        _getTokens(
            airdropAccounts[i].account,
            collection,
            airdropAccounts[i].amount,
            airdropAccounts[i].maxAmount
        );
    }
}

/// @notice Function to check for whitelisting
/// @dev Only Whitelist for users
/// @param collection Address NFT collection
/// @param proofs A set of proofs to confirm that an account is whitelisted
/// @param leaf Hash data account
function isWhitelist(
    address collection,
    bytes32[] calldata proofs,
    bytes32 leaf
) internal view virtual checkAddCollection(collection) returns (bool) {
    bytes32 merkleRoot = whitelistRoots[collection];

    return MerkleProof.verify(proofs, merkleRoot, leaf);
}
```

Recommendation:

As `isWhitelist` represents an `internal` function, we advise the modifier to be safely omitted from it optimizing the code's gas cost.

Alleviation:

The **`Airdrop::isWhitelist`** function was updated according to our recommendation, no longer applying redundant access control by omitting the **`Airdrop::checkAddCollection`** modifier.

APO-02C: Event Practicality Enhancements

Type	Severity	Location
Language Specific	Informational	Airdrop.sol:L213, L218

Description:

The GetTokens / UpdateWhiteListRoot events do not contain any indexed argument in their current implementation.

Example:

contracts/Airdrop.sol

SOL Copy

```sol
209 /// @notice Get Tokens event
210 /// @param collection Collection Address
211 /// @param account Account Address
212 /// @param amount The Number tokens
213 event GetTokens(address collection, address account, uint256 amount);
214
215 /// @notice Update WhiteList Root event
216 /// @param collection Collection Address
217 /// @param merkleRoot Hash merkle root
218 event UpdateWhiteListRoot(address collection, bytes32 merkleRoot);
```

Recommendation:

We advise them to introduce the indexed keyword for the collection member, aiding off-chain services in filtering events about a particular NFT collection as their queries would execute in less time and incur a smaller off-chain computational footprint.

Alleviation:

Both events have been updated, introducing the indexed keyword to the collection argument as advised.

APO-03C: Inefficient Administrative Mint Workflow

Type	Severity	Location
Gas Optimization	Informational	Airdrop.sol:L136-L141

Description:

The getTokensAdmin function is meant to circumvent the MerkleProof validation mechanism to directly mint a collection to a user, however, the function makes use of _getTokens which will still apply the relevant maximum amount checks.

Example:

contracts/Airdrop.sol

SOL Copy

```sol
/// @notice Function for issuing NFT collection tokens for free
/// @dev Only Admin
/// @param collection Address NFT collection
/// @param airdropAccounts Account Dataset
function getTokensAdmin(
    address collection,
    AirdropToken[] calldata airdropAccounts
) external onlyAdmin checkAddCollection(collection) {
    for (uint256 i = 0; i < airdropAccounts.length; i++) {
        _getTokens(
            airdropAccounts[i].account,
            collection,
            airdropAccounts[i].amount,
            airdropAccounts[i].maxAmount
        );
    }
}
```

Recommendation:

We advise the administrative mint workflow to not apply these checks and to directly mint the asset for the `_account`, simply emitting the `GetTokens` event in the process.

Alleviation:

The code was updated as advised, minting the `collection` directly to the target `account` and bypassing any potential limitations that are set by **`Airdrop::_getTokens`**.

APO-04C: Inefficient `mapping` Lookups

	rity	tion
Optimization	nformational	op.sol:L175, L181

Description:

The linked statements perform key-based lookup operations on `mapping` declarations from storage multiple times for the same key redundantly.

Example:

contracts/Airdrop.sol

SOL Copy

```sol
function _getTokens(
    address _account,
    address _collection,
    uint256 _amount,
    uint256 _maxAmount
) private {
    ERC721Collection erc721 = ERC721Collection(_collection);
    bytes32 accountHash = keccak256(abi.encodePacked(_account, _maxAmount));

    require(
        dropTokenAccounts[accountHash][_collection] + _amount <= _maxAmount,
        "This account has already received a free token."
    );

    erc721.mint(_account, _amount);

    dropTokenAccounts[accountHash][_collection] += _amount;

    emit GetTokens(_collection, _account, _amount);
}
```

Recommendation:

As the lookups internally perform an expensive `keccak256` operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the `mapping` in case of primitive types or holds a `storage` pointer to the `struct` contained.

Alleviation:

The interim `dropTokenAccounts[accountHash]` evaluation is now cached to a local variable and consequently utilized in the two referenced instances of the exhibit, optimizing the code's gas cost.

APO-05C: Loop Iterator Optimizations

	rity	tion
Optimization	nformational	op.sol:L135, L194

Description:

The linked `for` loops increment / decrement their iterator "safely" due to Solidity's built - in safe arithmetics(post - `0.8.X`).

Example:

contracts/Airdrop.sol

SOL Copy

```
135 for (uint256 i = 0; i < airdropAccounts.length; i++) {
```

Recommendation:

We advise the increment / decrement operations to be performed in an `unchecked` code block as the last statement within each `for` loop to optimize their execution cost.

Alleviation:

All iterator increment statements have been updated, incrementing the iterator within an `unchecked` block optimally.

BytesLibrary Code Style Findings

BLY-01C: Misleading Library Name

Type	Severity	Location
Code Style	Informational	BytesLibrary.sol:L4

Description:

The `BytesLibrary` name is misleading as the code of the `library` contains a `recover` cryptographic mechanism.

Example:

contracts/libs/BytesLibrary.sol

SOL Copy

```
4  library BytesLibrary {
5      function recover(
6          bytes32 message,
7          uint8 v,
8          bytes32 r,
9          bytes32 s
10     ) internal pure returns (address) {
11         return
12             ecrecover(
13                 keccak256(
14                     abi.encodePacked(
15                         "\x19Ethereum Signed Message:\n32",
16                         message
17                     )
18                 ),
19                 v,
20                 r,
21                 s
22             );
23     }
24 }
```

Recommendation:

We advise the `library` to be aptly renamed properly illustrating what its code contains.

Alleviation:

The relevant file of the exhibit has been removed from the codebase rendering it no longer applicable.

ERC20TransferProxy Code Style Findings

ERT-01C: Non-Standard Usage of Function Signature Literals

Type	Severity	Location
Code Style	Informational	ERC20TransferProxy.sol:L21, L41

Description:

The referenced statements construct a low-level call to the `addressToken` implementation representing either a `transferFrom(address,address,uint256)` or `transfer(address,uint256)` invocation, however, this is achieved via the usage of value literals for the signatures.

Example:

contracts/Proxies/ERC20TransferProxy.sol

SOL

Copy

```sol
/// @notice Calls transferFrom for ERC20 and fails on error.
/// @dev Can be called only by admin.
/// @param addressToken Token ERC20
/// @param from Account address from where to transfer
/// @param to Account address where to transfer
/// @param value Amount tokens ERC20
function erc20safeTransferFrom(
    address addressToken,
    address from,
    address to,
    uint256 value
) external onlyAdmin {
    (bool success, bytes memory data) = addressToken.call(
        abi.encodeWithSelector(0x23b872dd, from, to, value)
    );

    require(
        success && (data.length == 0 || abi.decode(data, (bool))),
        "TRANSFER_FROM_FAILED"
    );
}

/// @notice Calls transfer for ERC20 and fails on error.
/// @dev Can be called only by admin.
/// @param addressToken Token ERC20
/// @param to Account address where to transfer
/// @param value Amount tokens ERC20
function erc20safeTransfer(
    address addressToken,
    address to,
    uint256 value
) external onlyAdmin {
    (bool success, bytes memory data) = addressToken.call(
        abi.encodeWithSelector(0xa9059cbb, to, value)
    );

    require(
        success && (data.length == 0 || abi.decode(data, (bool))),
        "TRANSFER_FAILED"
    );
}
```

Recommendation:
We advise the IERC20 interface by OpenZeppelin to be imported to the codebase and the special selector accessor statement to be utilized on its functions in place of the value literal signatures (i.e. IERC20.transferFrom.selector), optimizing the legibility of the codebase and eliminating the potential for human error.

Alleviation:
The relevant function selectors from the IERC20 interface are now in use instead of the value literals, optimizing the code's legibility.

ERC721Collection Code Style Findings

ERN-01C: Loop Iterator Optimization

Type	Severity	Location
Gas Optimization	Informational	ERC721Collection.sol:L198

Description:
The linked for loop increments / decrements the iterator "safely" due to Solidity's built-in safe arithmetics(post - 0.8.X).

Example:

contracts/ERC721Collection.sol

SOL Copy

```
198 for (uint256 i = 0; i < amount; i++) {
```

Recommendation:
We advise the increment / decrement operation to be performed in an unchecked code block as the last statement within the for loop to optimize its execution cost.

Alleviation:
The referenced iterator has been optimized as advised, wrapping it in an unchecked code block during increments.

ERN-02C: Redundant Conditional Structure

Type	Severity	Location
Gas Optimization	Informational	ERC721Collection.sol:L217-L223, L225

Description:
The referenced conditional structure will evaluate a condition, return another conditional to the caller if it succeeds and true otherwise.

Example:

contracts/ERC721Collection.sol

SOL Copy

```
217 if (address(whitelistContractFilter) != address(0x0)) {
218     return
219         whitelistContractFilter.isApprovalContractAccount(
220             address(this),
221             account
222         );
223 }
224
225 return true;
```

Recommendation:

We advise the conditions to be yielded to the caller directly optimizing the code's gas cost by combining them in their correct format (i.e. `checkApproval` should yield `true` if the `whitelistContractFilter` is zero or if the `isApprovalContractAccount` succeeds).

Alleviation:

The conditional structure has been simplified to a direct `return` statement of a boolean evaluation as advised.

ExchangeDomain Code Style Findings
EDN-01C: Generic Typographic Mistake

Type	Severity	Location
Code Style	Informational	ExchangeDomain.sol:L6

Description:

The referenced line contains a typographical mistake (i.e. `private` variable without an underscore prefix) or generic documentational error (i.e. copy-paste) that should be corrected.

Example:

contracts/NFT-Marketplace/ExchangeDomain.sol

SOL Copy

```
6   /// @notice Describes all the structs that are used in exchnages.
```

Recommendation:

We advise this to be done so to enhance the legibility of the codebase.

Alleviation:

The typographic mistake has been corrected, alleviating this exhibit.

EDN-02C: Potential Data Structure Optimization

Type	Severity	Location
Gas Optimization	Informational	ExchangeDomain.sol:L15, L33, L35

Description:

The data structures of the exchange defined in `ExchangeDomain` can be optimized as they presently contain two data points that can be merged into one.

Example:

contracts/NFT-Marketplace/ExchangeDomain.sol

SOL

Copy

```sol
enum AssetType {
    ERC20,
    ERC721
}

struct Asset {
    address token;
    uint256 tokenId;
    AssetType assetType;
}

struct OrderKey {
    /* who signed the order */
    address owner;
    /* random number */
    uint256 salt;
    /* what has owner */
    Asset sellAsset;
    /* what wants owner */
    Asset buyAsset;
}

struct Order {
    OrderKey key;
    /* how much has owner (in wei, or UINT256_MAX if ERC-721) */
    uint256 selling;
    /* how much wants owner (in wei, or UINT256_MAX if ERC-721) */
    uint256 buying;
    /* fee. Represented as percents * 100 (100% - 10000. 1% - 100)*/
    uint256 fee;
}
```

Recommendation:

Presently, an `Order` struct contains two values indicating the "amount" of an asset that is being sold or bought, with NFTs being a special case in the `Asset` declaration whereby a `tokenId` is specified and the amount is expected to be equal to `type(uint256).max`. To avoid redundant data points, the `tokenId` member of `Asset` can be renamed to `tokenIdOrAmount`, rendering the `selling` and `buying` variables in the

`Order` redundant as an `AssetType` of `ERC20` would treat the `tokenIdOrAmount` variable as an `amount` whilst an `AssetType` of `ERC721` would treat the `tokenIdOrAmount` variable as a `tokenId`.

Alleviation:

The 10101 Art team has specified that they intend to use this data structure in future implementations to support other standards such as **EIP-1155** which would require both a token ID and an amount to be specified. As a result, we consider this exhibit nullified given that the code presents the most optimal data structure in light of these future adjustments.

ExchangeOrdersHolder Code Style Findings

EOH-01C: Suboptimal Struct Declaration Style

Type	Severity	Location
Code Style	Informational	ExchangeOrdersHolder.sol:L36

Description:

The linked declaration style of a struct is using index-based argument initialization.

Example:

contracts/NFT-Marketplace/ExchangeOrdersHolder.sol

SOL Copy

```
36  orders[key] = OrderParams(order.selling, order.buying, order.fee);
```

Recommendation:

We advise the key-value declaration format to be utilized instead, greatly increasing the legibility of the codebase.

Alleviation:

The key-value declaration style is now properly utilized in the referenced statement greatly increasing its legibility.

ExchangeState Code Style Findings

ESE-01C: Discrepant Key Encoding Mechanism

Type	Severity	Location
Standard Conformity	Informational	ExchangeState.sol:L49-L56

Description:

The key encoding mechanism in `ExchangeState::getCompletedKey` differs from the one employed by `ExchangeOrdersHolder::prepareKey` in both the mechanism used (`abi.encodePacked` vs `abi.encode`) and the order the arguments are present in the encodings.

Example:

contracts/NFT-Marketplace/ExchangeState.sol

SOL Copy

```sol
/// @notice Encode order key to use as the mapping key.
/// @param key - the `OrderKey` struct.
/// @return Encoded order key.
function getCompletedKey(ExchangeDomain.OrderKey memory key)
    public
    pure
    returns (bytes32)
{
    return
        keccak256(
            abi.encodePacked(
                key.owner,
                key.sellAsset.token,
                key.sellAsset.tokenId,
                key.buyAsset.token,
                key.buyAsset.tokenId,
                key.salt
            )
        );
}
```

Recommendation:
We advise the key generation mechanism to be streamlined, potentially in a library, to ensure that order keys generated within the system are consistent across modules.

Alleviation:
The encoding mechanism of both contracts has been relocated to an Encoding library which exposes a generateKey function that is in use throughout the system and ensures that the key generation mechanism is consistent. As such, we consider this exhibit fully alleviated.

HasSecondarySaleFees Code Style Findings
HSS-01C: Event Practicality Enhancement

Type	Severity	Location
Language Specific	Informational	HasSecondarySaleFees.sol:L8-L12

Description:
The SecondarySaleFees event does not contain any indexed argument in its current implementation.

Example:

contracts/NFT-Marketplace/HasSecondarySaleFees.sol

SOL Copy

```sol
event SecondarySaleFees(
    uint256 tokenId,
    address[] recipients,
    uint256[] bps
);
```

Recommendation:

We advise it to introduce the `indexed` keyword for the `tokenId` member, aiding off-chain services in filtering events about a particular NFT ID as their queries would execute in less time and incur a smaller off-chain computational footprint.

Alleviation:

The referenced event's `tokenId` argument has been set as `indexed`, optimizing off-chain filters utilizing it and alleviating this exhibit.

HSS-02C: Non-Standard Literal Definition of EIP-165 ID

Type	Severity	Location
Code Style	Informational	HasSecondarySaleFees.sol:L20

Description:

The referenced statement is accompanied by comments indicating how the interface ID for the `HasSecondarySaleFees` contract was generated, however, this is achieved via the usage of literals rather than code.

Example:

contracts/NFT-Marketplace/HasSecondarySaleFees.sol

SOL Copy

```
/// @title Abstract contract "Has Secondary Sale Fees"
abstract contract HasSecondarySaleFees is ERC165Storage {
    event SecondarySaleFees(
        uint256 tokenId,
        address[] recipients,
        uint256[] bps
    );

    /*
     * bytes4(keccak256('getFeeBps(uint256)')) == 0x0ebd4c7f
     * bytes4(keccak256('getFeeRecipients(uint256)')) == 0xb9c4d9fb
     *
     * => 0x0ebd4c7f ^ 0xb9c4d9fb == 0xb7799584
     */
    bytes4 private constant _INTERFACE_ID_FEES = 0xb7799584;

    constructor() {
        _registerInterface(_INTERFACE_ID_FEES);
    }

    function getFeeRecipients(uint256 id)
        public
        view
        virtual
        returns (address payable[] memory);

    function getFeeBps(uint256 id)
        public
        view
        virtual
        returns (uint256[] memory);
}
```

Recommendation:

We advise the statement and comments to be omitted and the functions of the HasSecondarySaleFees contract to be clearly defined in an inherited interface. Consequently, the interface can be imported to the codebase and its interfaceId can be extracted via the type statement (i.e. for an interface IHasSecondarySaleFees its ID can be extracted via type(IHasSecondarySaleFees).interfaceId).

Alleviation:

An IHasSecondarySaleFees file has been introduced to the codebase that defines the relevant functions of the contract and is now in use by the **HasSecondarySaleFees::constructor** in the syntax we advised, addressing this exhibit in full.

MarketMaker Code Style Findings

MMR-01C: Non-Standard Definition of Unitary Maximum

Type	Severity	Location
Code Style	Informational	MarketMaker.sol:L55

Description:

The referenced calculation is meant to represent the maximum of the uint256 data type, however, this is achieved via calculations rather than proper code syntax.

Example:

contracts/NFT-Marketplace/MarketMaker.sol

SOL Copy

```
55  uint256 private constant UINT256_MAX = 2**256 - 1;
```

Recommendation:

We advise the statement to be replaced by type(uint256).max optimizing the legibility of the code.

Alleviation:

The type(uint256).max syntax is now utilized by the code as advised.

MMR-02C: Non-Standard Literal Definition of EIP-165 ID

Type	Severity	Location
Code Style	Informational	MarketMaker.sol:L54

Description:

The referenced statement is meant to represent the interface ID of the HasSecondarySaleFees contract, however, this is achieved via the usage of a literal rather than code.

Example:

contracts/NFT-Marketplace/MarketMaker.sol

SOL Copy

```
54  bytes4 private constant _INTERFACE_ID_FEES = 0xb7799584;
```

Recommendation:

We advise the statement to be replaced akin to the homonym finding in the HasSecondarySaleFees contract.

Alleviation:
The IHasSecondarySaleFees interface defined for HSS-02C is utilized in the same fashion in this instance, alleviating this exhibit.

MMR-03C: Redundant Function Arguments

Type	Severity	Location
Gas Optimization	Informational	MarketMaker.sol:L317, L345-L349

Description:
The subFeeInBp function is always invoked with the same value and total argument.

Example:

contracts/NFT-Marketplace/MarketMaker.sol

SOL Copy

```
345 function subFeeInBp(
346     uint256 value,
347     uint256 total,
348     uint256 feeInBp
349 ) internal pure returns (uint256 newValue, uint256 realFee) {
350     return subFee(value, total.bp(feeInBp));
351 }
```

Recommendation:
We advise the function to be adjusted to accept a single argument instead, optimizing its gas cost.

Alleviation:
The **MarketMaker::subFeeInBp** function was updated according to our recommendation, merging the value and total arguments into the value argument as they were identical when used in the codebase.

MMR-04C: Redundant Numeric Enum Comparison

Type	Severity	Location
Code Style	Informational	MarketMaker.sol:L390

Description:
The referenced statement performs an enum comparison by casting them to unitary values redundantly as the AssetType declaration contains only two enum values.

Example:

contracts/NFT-Marketplace/MarketMaker.sol

SOL Copy

```sol
function getFeeSide(AssetType sellType, AssetType buyType)
    internal
    pure
    returns (FeeSide)
{
    if ((sellType == AssetType.ERC721) && (buyType == AssetType.ERC721)) {
        return FeeSide.NONE;
    }
    if (uint256(sellType) > uint256(buyType)) {
        return FeeSide.BUY;
    }
    return FeeSide.SELL;
}
```

Recommendation:

We advise the `enum` value to be evaluated directly as being equal to `AssetType.ERC20`, significantly increasing the legibility of the codebase.

Alleviation:

The `sellType` value is now properly utilized in a comparison as an `enum` instead of a unit, optimizing its legibility.

MMR-05C: Redundant Payable Address Casts

Type	Severity	Location
Code Style	Informational	MarketMaker.sol:L163-L164, L171-L172, L256-L257, L272, L297, L307

Description:

The linked statements all utilize the special `payable` sub-type of the `address` variable type redundantly.

Example:

contracts/NFT-Marketplace/MarketMaker.sol

SOL Copy

```sol
function transferWithFees(
    Asset memory firstType,
    uint256 value,
    address from,
    address payable to,
    uint256 fee
) internal {
    uint256 restValue = transferFeeToBeneficiary(
        firstType,
        from,
        value,
        fee
    );

    address payable toPayable = to;
    transfer(firstType, restValue, from, toPayable);
}
```

Recommendation:

We advise all `payable` casts and relevant usages in the referenced functions to be safely omitted from the codebase, optimizing its legibility.

Alleviation:

The `payable` attribute in use throughout the code of the referenced statements has been omitted, no longer requiring any casts to be performed and thus alleviating this exhibit indirectly.

MMR-06C: Variable Mutability Specifier (Immutable)

Type	Severity	Location
Gas Optimization	Informational	MarketMaker.sol:L86

Description:

The linked variable is assigned to only once during the contract's `constructor`.

Example:

contracts/NFT-Marketplace/MarketMaker.sol

SOL Copy

```sol
constructor(
    TransferProxy _transferProxy,
    ERC20TransferProxy _erc20TransferProxy,
    ExchangeState _state,
    ExchangeOrdersHolder _ordersHolder,
    address payable _beneficiary,
    address _aliveUntilSigner
) {
    transferProxy = _transferProxy;
    erc20TransferProxy = _erc20TransferProxy;
    state = _state;
    ordersHolder = _ordersHolder;
    beneficiary = _beneficiary;
    aliveUntilSigner = _aliveUntilSigner;
}
```

Recommendation:
We advise it to be set as `immutable` greatly optimizing its read-access gas cost.

Alleviation:
A `MarketMaker::setExchangeOrdersHolder` function has instead been declared, enabling the `ordersHolder` member to be adjusted and thus rendering this exhibit no longer applicable.

OwnableExt Code Style Findings
OET-01C: Inconsistent State Transition Restrictions

Type	Severity	Location
Code Style	Informational	OwnableExt.sol:L32-L34, L39-L45

Description:
The `OwnableExt` contract is meant to maintain an `admins` mapping of multiple users who are authorized in addition to the owner of the contract. In this mechanism, the `deleteAdmin` function behaves strictly and will only permit an administrator being removed only if they existed in the first place, however, the `addAdmin` function performs no check to validate whether the `_account` is already an administrator.

Example:

contracts/OwnableExt.sol

SOL Copy

```sol
/// @notice Function to add admin
/// @dev Only Owner
/// @param _account Address account
function addAdmin(address _account) external onlyOwner {
    admins[_account] = true;
}

/// @notice Function to remove admin
/// @dev Only Owner
/// @param _account Address account
function deleteAdmin(address _account)
    external
    onlyOwner
    checkExistAdmin(_account)
{
    delete admins[_account];
}
```

Recommendation:

We recommend the behaviour of the contract to be streamlined by either removing the checkExistAdmin evaluation from deleteAdmin or introducing a new check in addAdmin that ensures the _account being set as an administrator has not already been done so, the latter of which we advise.

Alleviation:

The code of both **OwnableExt::addAdmin** and **OwnableExt::deleteAdmin** was updated to properly maintain the list of admins by ensuring they either do not exist or do exist in each case respectively before applying the desired action.

Presale Code Style Findings

PEL-01C: Duplicate Application of Modifier

Type	Severity	Location
Gas Optimization	Informational	Presale.sol:L142

Description:

The checkAddCollection modifier is applied by the isWhitelist function which is in turn invoked in the getTokens and getTotalPriceCollection call-chains that both apply the checkAddCollection modifier.

Example:

contracts/Presale.sol

SOL Copy

```sol
function isWhitelist(
    address collection,
    bytes32[] calldata proofs,
    bytes32 leaf
) internal view override checkAddCollection(collection) returns (bool) {
    bool isWhiteilst = Airdrop.isWhitelist(collection, proofs, leaf);

    if (
        isWhiteilst &&
        (collectionInformations[collection].startWhitelistTimestamp <=
            block.timestamp &&
            block.timestamp <=
            collectionInformations[collection].stopWhitelistTimestamp)
    ) {
        return true;
    }

    return false;
}
```

Recommendation:

As `isWhitelist` represents an `internal` function, we advise the modifier to be safely omitted from it optimizing the code's gas cost.

Alleviation:

The **`Presale::isWhitelist`** function was updated according to our recommendation, no longer applying redundant access control by omitting the **`Airdrop::checkAddCollection`** modifier.

PEL-02C: Generic Typographic Mistake

Type	Severity	Location
Code Style	Informational	Presale.sol:L143

Description:

The referenced line contains a typographical mistake (i.e. `private` variable without an underscore prefix) or generic documentational error (i.e. copy-paste) that should be corrected.

Example:

contracts/Presale.sol

SOL Copy

```sol
143 bool isWhiteilst = Airdrop.isWhitelist(collection, proofs, leaf);
```

Recommendation:
We advise this to be done so to enhance the legibility of the codebase.

Alleviation:
The typographic mistake is no longer present in the codebase, rendering this exhibit alleviated.

PEL-03C: Inefficient Data Pointers

Type	Severity	Location
Gas Optimization	Informational	Presale.sol:L233, L292

Description:
The referenced statements perform a `memory` assignment of the `Collection` struct whilst only few of its members (`whitelistPrice` & `publicPrice` / `erc20Address`) are utilized within their respective functions.

Example:

contracts/Presale.sol

SOL Copy

```sol
Collection memory collectionInformation = collectionInformations[
    _collection
];

uint256 whitelistTokenAmount = getWhitelistAmount(
    _collection,
    _accountHash,
    _proofs,
    _amount,
    _maxAmountWhiteList
);

totalPrice =
    collectionInformation.whitelistPrice *
    whitelistTokenAmount +
    (collectionInformation.publicPrice *
        (amount - whitelistTokenAmount));

return totalPrice;
```

Recommendation:

We advise the assignments to be set as storage ones instead, optimizing each code's gas cost significantly.

Alleviation:

Only the latter of the two referenced declarations was optimized, rendering this exhibit partially alleviated.

PEL-04C: Inefficient mapping Lookups

Type	Severity	Location
Gas Optimization	Informational	Presale.sol:L147, L150, L214, L216, L241, L243, L245

Description:

The linked statements perform key-based lookup operations on mapping declarations from storage multiple times for the same key redundantly.

Example:

contracts/Presale.sol

SOL Copy

```sol
145 if (
146     isWhiteilst &&
147     (collectionInformations[collection].startWhitelistTimestamp <=
148         block.timestamp &&
149         block.timestamp <=
150         collectionInformations[collection].stopWhitelistTimestamp)
151 ) {
```

Recommendation:

As the lookups internally perform an expensive keccak256 operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the mapping in case of primitive types or holds a storage pointer to the struct contained.

Alleviation:

Only the highlighted section of the referenced declarations has been optimized, rendering this exhibit partially alleviated.

PEL-05C: Inexistent Specification of Override

Type	Severity	Location
Code Style	Informational	Presale.sol:L64

Description:

The referenced function overrides the parent implementation Airdrop::addCollection yet does not specify it explicitly.

Example:

contracts/Presale.sol

SOL Copy

```sol
55  function addCollection(
56      address collection,
57      uint256 whitelistPrice,
58      uint256 publicPrice,
59      uint256 startWhitelistTimestamp,
60      uint256 startPublicTimestamp,
61      uint256 stopWhitelistTimestamp,
62      uint256 stopTimestamp,
63      address erc20Address
64  ) external onlyAdmin checkRemoveCollection(collection) {
```

Recommendation:

We advise the override keyword to be properly introduced to the function declaration, optimizing the legibility of the codebase.

Alleviation:
The 10101 Art team has opted to not apply a remediation for this exhibit instead acknowledging it.

PEL-06C: Loop Iterator Optimization

Type	Severity	Location
Gas Optimization	Informational	Presale.sol:L269

Description:
The linked `for` loop increments / decrements the iterator "safely" due to Solidity's built-in safe arithmetics(post - `0.8.X`).

Example:

contracts/Presale.sol

SOL Copy

```
269 for (uint256 i = 0; i < accountsDrop.length; i++) {
```

Recommendation:
We advise the increment / decrement operation to be performed in an `unchecked` code block as the last statement within the `for` loop to optimize its execution cost.

Alleviation:
The referenced iterator has been optimized as advised, wrapping it in an `unchecked` code block during increments.

PEL-07C: Redundant Conditional Structure

Type	Severity	Location
Gas Optimization	Informational	Presale.sol:L145-L153, L155

Description:
The referenced conditional structure will evaluate a condition, return `true` to the caller if it succeeds and `false` otherwise.

Example:

contracts/Presale.sol

SOL Copy

```sol
145 if (
146     isWhiteilst &&
147     (collectionInformations[collection].startWhitelistTimestamp <=
148         block.timestamp &&
149         block.timestamp <=
150         collectionInformations[collection].stopWhitelistTimestamp)
151 ) {
152     return true;
153 }
154
155 return false;
```

Recommendation:
We advise the condition to be yielded to the caller directly optimizing the code's gas cost.

Alleviation:
The conditional structure has been simplified to a direct `return` statement of a boolean evaluation as advised.

SafeMath Code Style Findings
SMH-01C: Incorrect Usage of Dependency

Type	Severity	Location
Gas Optimization	Informational	SafeMath.sol:L2, L17

Description:
The `SafeMath` dependency the codebase utilizes is meant for `pragma` versions of `0.7.X` and below as it does not take into account the built-in safe arithmetics that are toggled on by default in `pragma` versions `0.8.0` and up.

Example:

contracts/libs/SafeMath.sol

SOL Copy

```sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, '"SafeMath: addition overflow"');

        return c;
    }
```

Recommendation:

We advise the SafeMath library to be omitted entirely from the codebase as it is no longer necessary and incurs extra gas cost at no benefit.

Alleviation:

The relevant file of the exhibit has been removed from the codebase rendering it no longer applicable.

WhitelistContractFilter Code Style Findings
WCF-01C: Inefficient mapping Lookups

Type	Severity	Location
Gas Optimization	Informational	WhitelistContractFilter.sol:L97-L99, L122-L124

Description:

The linked statements perform key-based lookup operations on mapping declarations from storage multiple times for the same key redundantly.

Example:

contracts/WhitelistContractFilter.sol

SOL Copy

```sol
/// @notice Function batch add filter element for element private WhitelistCon
/// @dev Public function. Only Admin.
/// @param filters Array struct contractAccountApprove (element -> filters);
function addFilterPrivateBatch(FilterBatch[] calldata filters)
    external
    onlyAdmin
{
    for (uint256 i = 0; i < filters.length; i++) {
        FilterBatch memory filter = filters[i];

        for (uint256 j = 0; j < filter.filters.length; j++) {
            if (!isContract(filter.filters[j])) continue;

            privateWhitelistContract[filter.element][
                filter.filters[j]
            ] = true;

            emit AddingApproveContractAccount(
                filter.element,
                filter.filters[j]
            );
        }
    }
}

/// @notice Function batch remove filter element for element private Whitelist
/// @dev Public function. Only Admin
/// @param filters Array struct contractAccountApprove (element -> filters);
function removeFilterPrivateBatch(FilterBatch[] calldata filters)
    external
    onlyAdmin
{
    for (uint256 i = 0; i < filters.length; i++) {
        FilterBatch memory filter = filters[i];

        for (uint256 j = 0; j < filter.filters.length; j++) {
            if (!isContract(filter.filters[j])) continue;

            privateWhitelistContract[filter.element][
                filter.filters[j]
            ] = false;

            emit RemovingApproveContractAccount(
                filter.element,
                filter.filters[j]
            );
        }
    }
```

Recommendation:
As the lookups internally perform an expensive keccak256 operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the mapping in case of primitive types or holds a storage pointer to the struct contained.

Alleviation:
While both instances contain a local variable that points to the relevant mapping entry, they do so inefficiently as the variable is declared within the for loop bodies. We advise the declarations to be relocated outside the for loop that iterates through each inner-level filter of a higher-level filter, optimizing the code significantly.

WCF-02C: Loop Iterator Optimizations

Type	Severity	Location
Gas Optimization	Informational	WhitelistContractFilter.sol:L91, L94, L116, L119, L141, L157

Description:
The linked for loops increment / decrement their iterator "safely" due to Solidity's built - in safe arithmetics(post - 0.8.X).

Example:

contracts/WhitelistContractFilter.sol

SOL Copy

```
91  for (uint256 i = 0; i < filters.length; i++) {
```

Recommendation:
We advise the increment / decrement operations to be performed in an unchecked code block as the last statement within each for loop to optimize their execution cost.

Alleviation:
All iterator increment statements have been updated, incrementing each iterator within an unchecked block optimally.

WCF-03C: Redundant Duplication of Code

Type	Severity	Location
Code Style	Informational	WhitelistContractFilter.sol:L204-L206

Description:
The referenced code is meant to be implemented by the Address contract present in the codebase.

Example:

contracts/WhitelistContractFilter.sol

SOL Copy

```
204 function isContract(address account) internal view returns (bool) {
205     return account.code.length > 0;
206 }
```

Recommendation:
We advise it to be properly imported into the code and its `isContract` function to be utilized, minimizing code duplication and inconsistencies.

Alleviation:
The **`WhitelistContractFilter::isContract`** function has been omitted from the contract and is no longer in use, rendering this exhibit indirectly alleviated.

WCF-04C: Simplification of Ternary Operators

	rity	tion
Optimization	formational	elistContractFilter.sol:L184-L187, L199-L20

Description:
The ternary operators in use by the codebase are redundant and can be omitted by adjusting the conditionals they are used in.

Example:

contracts/WhitelistContractFilter.sol

SOL Copy

```sol
/// @notice Function check filter element for WhitelistContract (and if contra
/// @dev Public function. For read
/// @param element address element for WhitelistContract
/// @param contractAccount address filter for element
function isApprovalContractAccount(address element, address contractAccount)
    public
    view
    returns (bool)
{
    return
        activeFilter && isContract(contractAccount)
            ? isExistApprovalContractAccount(element, contractAccount)
            : true;
}

/// @notice Function check filter element for contractAccount element Whitelis
/// @dev Private function. For read
/// @param element address element for WhitelistContract (0x0 address - public
/// @param contractAccount address filter for element
function isExistApprovalContractAccount(
    address element,
    address contractAccount
) private view returns (bool) {
    return
        publicWhitelistContract[contractAccount]
            ? true
            : privateWhitelistContract[element][contractAccount];
}
```

Recommendation:

We advise this to be done so, optimizing the code's legibility as well as gas cost. As an example, the ternary operator in `isApprovalContractAccount` can be adjusted to `!activeFilter || !isContract(contractAccount) || isExistApprovalContractAccount(element, contractAccount)` as it is equivalent and clearly depicts that if the filter is not active or the `contractAccount` does not represent a contract no check needs to be performed.

Alleviation:

The redundant ternary operators have been simplified to a single logical clause, optimizing the code's legibility as well as gas cost.

Finding Types

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted if blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Circom language boasts that discerns it from other conventional programming languages.

Curve Specific

Circom defaults to using the BN128 scalar field (a 254-bit prime field), but it also supports BSL12-381 (which has a 255-bit scalar field) and Goldilocks (with a 64-bit scalar field). However, since there are no constants denoting either the prime or the prime size in bits available in the Circom language, some Circomlib templates like `Sign` (which returns the sign of the input signal), and `AliasCheck` (used by the strict versions of `Num2Bits` and `Bits2Num`), hardcode either the BN128 prime size or some other constant related to BN128. Using these circuits with a custom prime may thus lead to unexpected results and should be avoided.

Code Style

In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a local-level variable contains the same name as a toplevel variable in the circuit.

Mathematical Operations

This category is used when a mathematical issue is identified. This implies an issue with the implementation of a calculation compared to the specifications.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

Privacy Concern

This category is used when information that is meant to be kept private is made public in some way.

Proof Concern

Under-constrained signals are one of the most common issues in zero-knowledge circuits. Issues with proof generation fall under this category.

Disclaimer

IMPORTANT TERMS & CONDITIONS REGARDING OUR SECURITY AUDITS/REVIEWS/REPORTS AND ALL PUBLIC/PRIVATE CONTENT/DELIVERABLES

Omniscia ("Omniscia") has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes. In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. Omniscia and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

This audit report shall not be printed, saved, disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Omniscia.

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. Omniscia is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. Omniscia's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

Omniscia in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will Omniscia, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the “client”) with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

Omniscia will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.
Omniscia will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.
For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.

EXTERNAL SOURCES

Source Code