OMNISCIA

May 9, 2023

SMART CONTRACT AUDIT REPORT

Gnosis Guild - Zodiac Modifier Roles


omniscia.io


info@omniscia.io


Online report: gnosis-guild-zodiac-modifier-roles

Zodiac Modifier Roles Security Audit

Audit Revisions

Commit Hash	Date	Revision Hash
23b782f781	May 9th 2023	916e3ac17f

Audit Overview

We were tasked with performing an audit of the Gnosis Guild codebase and in particular their specialized access control module meant to enforce access control of various Gnosis-related modules.

The system enables users to enforce a significantly granular level of access control that differs and adapts to each transaction's target as well as function selector.

Over the course of the audit, we identified contradictory behaviour in the `XOR` operator supported by the conditions in the system.

We advise the Gnosis Guild 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 Gnosis Guild 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 Gnosis Guild and have identified that all exhibits have been adequately dealt with no outstanding issues remaining in the report.

Contracts Assessed

Files in Scope	Repository	Commit(s)
BufferPacker.sol (BPR)	zodiac-modifier-roles	a9f65e8f05, 23b782f781
Core.sol (CER)	zodiac-modifier-roles	a9f65e8f05, 23b782f781
Consumptions.sol (CSN)	zodiac-modifier-roles	a9f65e8f05, 23b782f781
Decoder.sol (DRE)	zodiac-modifier-roles	a9f65e8f05, 23b782f781
Integrity.sol (IYT)	zodiac-modifier-roles	a9f65e8f05, 23b782f781
MultiSendUnwrapper.sol (MSU)	zodiac-modifier-roles	a9f65e8f05, 23b782f781
Packer.sol (PRE)	zodiac-modifier-roles	a9f65e8f05, 23b782f781
Periphery.sol (PYR)	zodiac-modifier-roles	a9f65e8f05, 23b782f781
PermissionLoader.sol (PLR)	zodiac-modifier-roles	a9f65e8f05, 23b782f781
PermissionBuilder.sol (PBR)	zodiac-modifier-roles	a9f65e8f05, 23b782f781
PermissionChecker.sol (PCR)	zodiac-modifier-roles	a9f65e8f05, 23b782f781
PermissionTracker.sol (PTR)	zodiac-modifier-roles	a9f65e8f05, 23b782f781
Roles.sol (RSE)	zodiac-modifier-roles	a9f65e8f05, 23b782f781

Files in Scope	Repository	Commit(s)
Types.sol (TSE)	zodiac-modifier-roles	a9f65e8f05, 23b782f781
Types.sol (TSP)	zodiac-modifier-roles	a9f65e8f05, 23b782f781
Topology.sol (TYG)	zodiac-modifier-roles	a9f65e8f05, 23b782f781
WriteOnce.sol (WOE)	zodiac-modifier-roles	a9f65e8f05, 23b782f781

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	0	0	0	0
Informational	27	14	0	13
Minor	2	2	0	0
Medium	2	2	0	0
Major	0	0	0	0

During the audit, we filtered and validated a total of **3 findings utilizing static analysis** tools as well as identified a total of **28 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 TypeScript.

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 `0.8.17` based on the version specified within the `hardhat.config.ts` file.

The project contains discrepancies with regards to the Solidity version used as the `pragma` statements of the contracts are open-ended (`>=0.8.17`).

We advise them to be locked to `0.8.17` (`=0.8.17`), 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 **173 potential issues** within the codebase of which **168 were ruled out to be false positives** or negligible findings.

The remaining **5 issues** were validated and grouped and formalized into the **3 exhibits** that follow:

ID	Severity	Addressed	Title
IYT-01S	Informational	Acknowledged	Data Location Optimization
PCR-01S	Informational	Yes	Literal Equality of `bool` Variable
RSE-01S	Informational	Acknowledged	Inexistent Sanitization of Input Addresses

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Gnosis Guild's specialized access control `Roles` module.

As the project at hand implements a customized access control mechanism with specialized encoding mechanisms and a multi-purpose structure, intricate care was put into ensuring that the **access flow 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 a vulnerability** within the system's `XOR` logical condition implementation 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 hard-coded offsets in use throughout the codebase.

A total of **28 findings** were identified over the course of the manual review of which **6 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
IYT-01M	Minor	Yes	Weak Validation of Children Counts
IYT-02M	Minor	Yes	Weak Validation of Comparator Values
PCR-01M	Medium	Yes	Discrepant XOR Behaviour
TYG-01M	Medium	Yes	Potentially Invalidated Assumption
TSE-01M	Informational	Nullified	Potentially Improper Function Mutability
WOE-01M	Informational	Acknowledged	Unsafe Casting Operation

Code Style

During the manual portion of the audit, we identified **22 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
BPR-01C	Informational	Acknowledged	Loop Iterator Optimization
CSN-01C	Informational	Yes	Inefficient Lookup Operation of Consumptions
IYT-01C	Informational	Yes	Content Validation Grouping
IYT-02C	Informational	Acknowledged	Loop Iterator Optimizations
IYT-03C	Informational	Yes	Redundant Validation of Sub-Type Tree
MSU-01C	Informational	Yes	Deprecated `if-revert` Pattern
MSU-02C	Informational	Yes	Non-Standard Definition of Function Signature
PRE-01C	Informational	Acknowledged	Loop Iterator Optimizations
PRE-02C	Informational	Yes	Optimization of Array Iteration
PYR-01C	Informational	Acknowledged	Redundant Code Duplication
PBR-01C	Informational	Yes	Suboptimal Struct Declaration Styles
PCR-01C	Informational	Nullified	Inefficient Iteration of Child Conditions
PCR-02C	Informational	Acknowledged	Optimization of Data Extraction
PTR-01C	Informational	Yes	Inefficient `mapping` Lookups
RSE-01C	Informational	Yes	Inefficient Transfer of Ownership
RSE-02C	Informational	Acknowledged	Loop Iterator Optimization

ID	Severity	Addressed	Title
TYG-01C	Informational	Acknowledged	Ineffectual Usage of Safe Arithmetics
TYG-02C	Informational	Yes	Loop Iterator Optimization
TSP-01C	Informational	Yes	Incorrect Grouping of `EqualToAvatar` Operation (Documentation)
TSE-01C	Informational	Acknowledged	Multiple Top-Level Declarations
TSP-02C	Informational	Acknowledged	Suboptimal Documentation of Enum
WOE-01C	Informational	Acknowledged	Multiple Top-Level Declarations

Integrity Static Analysis Findings

IYT-01S: Data Location Optimization

Type	Severity	Location
Gas Optimization	Informational	Integrity.sol:L31

Description:

The linked input argument is set as `memory` in an `external` function.

Example:

packages/evm/contracts/Integrity.sol

SOL

```sol
31  function enforce(ConditionFlat[] memory conditions) external pure {
```

Recommendation:

We advise it to be set as `calldata` optimizing its read-access gas cost. In order for the optimization to lead to a gas reduction, the code would need to be refactored to adapt to the new `calldata` type argument (i.e. the `Integrity::_content` function would also need to accept a `calldata` argument).

Alleviation:

The Gnosis Guild team has opted to not apply this optimization to the codebase to retain the present code's legibility.

PermissionChecker Static Analysis Findings

PCR-01S: Literal Equality of `bool` Variable

Type	Severity	Location
Gas Optimization	Informational	PermissionChecker.sol:L690

Description:

The linked `bool` comparison is performed between a variable and a `bool` literal.

Example:

packages/evm/contracts/PermissionChecker.sol

SOL

```
690 assert(found == true);
```

Recommendation:

We advise the `bool` variable to be utilized directly either in its negated (`!`) or original form.

Alleviation:

The literal `bool` comparison was replaced by the `found` variable's evaluation directly, optimizing the `assert` condition's evaluation cost.

Roles Static Analysis Findings

RSE-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	Roles.sol:L43-L46

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:

packages/evm/contracts/Roles.sol

SOL

```sol
43  constructor(address _owner, address _avatar, address _target) {
44      bytes memory initParams = abi.encode(_owner, _avatar, _target);
45      setUp(initParams);
46  }
```

Recommendation:

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

Alleviation:

The Gnosis Guild team evaluated this exhibit but has opted not to apply a remediation for it as an invalid `address` can be of many different values.

We would like to note that the reason the zero-address is protected in particular is that off-chain software will usually use "empty" values (i.e. `0x00..00`) when malfunctioning, hence the recommended course of

action.

In any case, we consider this exhibit safely acknowledged as it is trivial to re-deploy the contracts should the Gnosis Guild make a mistake.

Integrity Manual Review Findings

IYT-01M: Weak Validation of Children Counts

Type	Severity	Location
Input Sanitization	Minor	Integrity.sol:L119-L141

Description:

The referenced `for` loop within `Integrity::_topology` does not adequately validate multiple operators as the children count is validated solely for a subset of operators.

Impact:

The code currently permits malformed conditions to be defined that will misbehave.

Example:

packages/evm/contracts/Integrity.sol

SOL

```sol
for (uint256 i = 0; i < conditions.length; i++) {
    ConditionFlat memory condition = conditions[i];
    if (
        condition.paramType == ParameterType.Array &&
        childrenBounds[i].length == 0
    ) {
        revert UnsuitableChildrenCount(i);
    }
    if (
        (condition.operator == Operator.ArraySome ||
            condition.operator == Operator.ArrayEvery) &&
        childrenBounds[i].length != 1
    ) {
        revert UnsuitableChildrenCount(i);
    }

    if (
        condition.operator == Operator.ArraySubset &&
        childrenBounds[i].length > 256
```

```
138     ) {
139         revert UnsuitableChildrenCount(i);
140     }
141 }
```

Recommendation:

We advise the referenced code to be expanded, supporting validation of the missing operator types (i.e. `EqualToAvatar`, `Matches`, `GreaterThan`, `LessThan`, etc.) to ensure that they either have at least one child (i.e. in the case of `Matches`) or they do not have any children (i.e. in the case of `EqualToAvatar`).

Alleviation:

The **`Integrity::_topology`** function has been renamed to `Integrity::_tree` and now properly validates the children count of all `ParameterType` as well as potential operator sub-categories, alleviating this exhibit in full.

IYT-02M: Weak Validation of Comparator Values

Type	Severity	Location
Input Sanitization	Minor	Integrity.sol:L84-L100

Description:

The referenced `for` loop within `Integrity::_topology` does not adequately validate multiple operators as the comparator value (`compValue`) is validated solely for the logical subset of operators.

Impact:

The code currently permits malformed conditions to be defined that will misbehave.

Example:

packages/evm/contracts/Integrity.sol

SOL

```sol
// check at least 1 child for Logical
for (uint256 i = 0; i < conditions.length; i++) {
    ConditionFlat memory condition = conditions[i];
    if (
        (condition.operator >= Operator.And &&
            condition.operator <= Operator.Xor)
    ) {
        // must have no compValue
        if (condition.compValue.length != 0) {
            revert UnsuitableCompValue(i);
        }

        // must have at least one child
        if (childrenBounds[i].length == 0) {
            revert UnsuitableChildrenCount(i);
        }
    }
}
```

Recommendation:

We advise the referenced code to be expanded, supporting validation of the missing operator types (i.e. `EqualToAvatar`, `WithinAllowance` etc.) that shouldn't have a comparator value defined as well as operator types like `Custom` which must have a comparator value defined.

Alleviation:

The `compValue` entry of a particular condition is no longer validated in `Integirty::_topology` (now renamed to `Integrity::_tree`) and has been relocated to the `Integrity::_node` function. The revised code sanitizes the `compValue` of a condition based on the exact `Operator` type, ensuring that the `compValue` of each condition is either adequately present or absent.

As such, we consider this exhibit fully alleviated.

PermissionChecker Manual Review Findings

PCR-01M: Discrepant XOR Behaviour

Type	Severity	Location
Logical Fault	Medium	PermissionChecker.sol:L431

Description:

The logical `XOR` operator implemented by the `PermissionChecker` deviates from how the `XOR` operation would behave if applied to each individual child of the condition. In detail, a `XOR` sequence of `1 XOR 1 XOR 1` would yield a final output of `1`, however, a **`PermissionChecker::_xor`** operation of such child conditions would yield a final output of `0`.

Impact:

Conditions that are meant to utilize the `XOR` operator would behave differently than the traditional logical `XOR` operator, potentially causing users to misuse it.

Example:

packages/evm/contracts/PermissionChecker.sol

SOL

```sol
406 function _xor(
407     uint256 value,
408     bytes calldata data,
409     Condition memory condition,
410     ParameterPayload memory payload,
411     Consumption[] memory consumptions
412 ) private pure returns (Status, Result memory result) {
413     uint256 okCount;
414     unchecked {
415         for (uint256 i; i < condition.children.length; ++i) {
416             (Status status, Result memory _result) = _walk(
417                 value,
418                 data,
419                 condition.children[i],
420                 payload,
```

```
421                 consumptions
422             );
423             if (status == Status.Ok) {
424                 result = _result;
425                 okCount = okCount + 1;
426             }
427         }
428     }
429
430     return
431         okCount == 1
432             ? (Status.Ok, result)
433             : (
434                 Status.XorViolation,
435                 Result({consumptions: consumptions, info: 0})
436             );
437 }
```

Recommendation:

We advise this behaviour to be clearly documented in the `PermissionChecker` and if the deviation is undesirable, the referenced conditional to apply a modulo operation (`%`) to `okCount` with `2` imitating the behaviour of sequential `XOR` operations as in a real logical circuit.

Alleviation:

The Gnosis Guild team evaluated this exhibit, accepted that the present `XOR` behaviour is discrepant, and ultimately opted to remove this type of functionality from the logical subset of operators entirely. As such, we consider this exhibit alleviated.

Topology Manual Review Findings

TYG-01M: Potentially Invalidated Assumption

Type	Severity	Location
Logical Fault	Medium	Topology.sol:L34

Description:

The `Topology::childrenBounds` function assumes that the first member of the array will be the root (i.e. with a `parent` pointing to itself), however, this trait is not guaranteed by the sanitization performed at `Integrity::enforce`.

In detail, the `Integrity::_topology` function ensures that the array is in ascending order **but not in a strictly ascending order**, permitting equalities between members. As such, it is possible for a `conditions` array whereby the "root" is located at the `nth` element to exist as long as all the preceding elements have a `parent` equal to the `nth` element.

In such a case, the topology evaluated would be incorrect as the parent count would "reset" and not count the preceding elements while bounds will include the root itself, potentially causing an infinite recursion loop to manifest.

Impact:

The topology generated for a valid condition list that contains a root within it would be incorrect, causing multiple levels of validation throughout the code to fail proper execution.

Example:

packages/evm/contracts/Topology.sol

SOL

```sol
function childrenBounds(
    ConditionFlat[] memory conditions
) internal pure returns (Bounds[] memory result) {
    uint256 count = conditions.length;
    assert(count > 0);

```

```
29      unchecked {
30          // parents are breadth-first
31          result = new Bounds[](count);
32          result[0].start = type(uint256).max;
33
34          // first item is the root
35          for (uint256 i = 1; i < count; ++i) {
36              result[i].start = type(uint256).max;
37              Bounds memory parentBounds = result[conditions[i].parent];
38              if (parentBounds.start == type(uint256).max) {
39                  parentBounds.start = i;
40              }
41              parentBounds.end = i + 1;
42              parentBounds.length = parentBounds.end - parentBounds.start;
43          }
44      }
45  }
```

Recommendation:

We advise either the `Integrity::_root` function to be updated to ensure the root is present solely in the first element or the `Topology::childrenBounds` function to be adjusted to accommodate for a case whereby the root is located within the array, either of which we consider an adequate resolution to this exhibit and the former of which we advise.

Alleviation:

The code of `Integrity::enforce` and in detail `Integrity::_root` has been updated to mandate a single parent located at the first index of the `conditions` array thus alleviating this exhibit in full.

Types Manual Review Findings

TSE-01M: Potentially Improper Function Mutability

Type	Severity	Location
Standard Conformity	Informational	Types.sol:L31

Description:

The `ICustomCondition::check` function as defined in the `Types` is specified as `pure`, however, it should be possible to read its state (i.e. be specified as `view`).

Example:

packages/evm/contracts/adapters/Types.sol

SOL

```sol
interface ICustomCondition {
    function check(
        uint256 value,
        bytes calldata data,
        uint256 location,
        uint256 size,
        bytes12 extra
    ) external pure returns (bool success, bytes32 reason);
}
```

Recommendation:

We advise it to be set to `view` as in the current compiler utilized the `pure` assumption is also not enforced (i.e. a `pure` function can read the state).

Alleviation:

The Gnosis Guild team considered this exhibit and opted to retain the `pure` attribute as it better illustrates the custom checker's semantics. As such, we consider this exhibit nullified given that `pure` is the mutability the Gnosis Guild team wishes custom checkers to honour.

WriteOnce Manual Review Findings

WOE-01M: Unsafe Casting Operation

Type	Severity	Location
Input Sanitization	Informational	WriteOnce.sol:L110

Description:

The `WriteOnce::creationBytecodeFor` function will cast the input's `length` attribute plus `1` to a `uint32` data type without properly evaluating that the `length` satisfies the size limitation of the `PUSH4` operation code in use by the implementation.

Impact:

A `data` payload whose size exceeds the `type(uint32).max` limitation will successfully execute but truncate the `data` whenever it is read.

Example:

packages/evm/contracts/WriteOnce.sol

SOL

```
93  function creationBytecodeFor(
94      bytes memory data
95  ) private pure returns (bytes memory) {
96      /*
97    0x00    0x63         0x63XXXXXX  PUSH4 _code.length  size
98    0x01    0x80         0x80        DUP1                size size
99    0x02    0x60         0x600e      PUSH1 14            14 size size
100   0x03    0x60         0x6000      PUSH1 00            0 14 size size
101   0x04    0x39         0x39        CODECOPY            size
102   0x05    0x60         0x6000      PUSH1 00            0 size
103   0x06    0xf3         0xf3        RETURN
104   <CODE>
105 */
106
107     return
108         abi.encodePacked(
```

```
            hex"63",
            uint32(data.length + 1),
            hex"80_60_0E_60_00_39_60_00_F3",
            // Append 00 to data so contract can't be called
            hex"00",
            data
        );
}
```

Recommendation:

We advise the `data.length + 1` value to be ensured as fitting within an unsigned integer of 32-bits by ensuring that it is less-than-or-equal-to `type(uint32).max`. To note, Solidity's built-in safe arithmetic in post-`0.8.X` versions does not guarantee casting operations which need to be manually guarded.

Alleviation:

The Gnosis Guild team evaluated this exhibit but has opted not to apply a remediation for it as they do not foresee a production use case where an exploit would manifest as a result of this misbehaviour.

As such, we consider this exhibit safely acknowledged.

BufferPacker Code Style Findings

BPR-01C: Loop Iterator Optimization

Type	Severity	Location
Gas Optimization	Informational	BufferPacker.sol:L43

Description:

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

Example:

packages/evm/contracts/packers/BufferPacker.sol

SOL

```
43  for (uint256 i; i < count; ++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 Gnosis Guild team considered this exhibit but has opted not to apply it to avoid broad usage of the `unchecked` paradigm. As such, we consider this exhibit acknowledged.

Consumptions Code Style Findings

CSN-01C: Inefficient Lookup Operation of Consumptions

Type	Severity	Location
Gas Optimization	Informational	Consumptions.sol:L64

Description:

The `Consumptions::find` function that is invoked within `Consumptions::merge` is invoked so with a `result` array that contains multiple zeroed out entries that are iterated redundantly.

Example:

packages/evm/contracts/Consumptions.sol

SOL

```sol
46  function merge(
47      Consumption[] memory c1,
48      Consumption[] memory c2
49  ) internal pure returns (Consumption[] memory result) {
50      if (c1.length == 0) return c2;
51      if (c2.length == 0) return c1;
52
53      result = new Consumption[](c1.length + c2.length);
54
55      uint256 length = c1.length;
56      unchecked {
57          for (uint256 i; i < length; ++i) {
58              result[i].allowanceKey = c1[i].allowanceKey;
59              result[i].balance = c1[i].balance;
60              result[i].consumed = c1[i].consumed;
61          }
62
63          for (uint256 i; i < c2.length; ++i) {
64              (uint256 index, bool found) = find(result, c2[i].allowanceKey);
65              if (found) {
66                  result[index].consumed += c2[i].consumed;
67              } else {
68                  result[length].allowanceKey = c2[i].allowanceKey;
```

```
69                  result[length].balance = c2[i].balance;
70                  result[length].consumed = c2[i].consumed;
71                  length++;
72              }
73          }
74      }
```

Recommendation:

As the `result` array contains all `c1` elements at the same indexes, the **`Consumptions::find`** function could accept the `c1` array as an input and use the same index to operate on the `result` array.

To note, the contract itself assumes that `consumptions` do not contain duplicate `allowanceKey` entries by themselves rendering this approach possible.

Alleviation:

The input of the referenced **`Consumptions::find`** function invocation statement has been adjusted to the `c1` array as advised, optimizing the lookup operation significantly and thus alleviating this exhibit.

Integrity Code Style Findings

IYT-01C: Content Validation Grouping

Type	Severity	Location
Gas Optimization	Informational	Integrity.sol:L153-L169, L177-L189, L205-L233

Description:

The referenced `if-else-if` clauses can be grouped into a single conditional as they validate multiple operators of the same family (i.e. logical ones, array-based etc.).

Example:

packages/evm/contracts/Integrity.sol

SOL

```sol
144 function _content(
145     ConditionFlat memory condition,
146     uint256 index
147 ) private pure {
148     Operator operator = condition.operator;
149     ParameterType paramType = condition.paramType;
150     bytes memory compValue = condition.compValue;
151     if (operator == Operator.Pass) {
152         return;
153     } else if (operator == Operator.And) {
154         if (paramType != ParameterType.None) {
155             revert UnsuitableParameterType(index);
156         }
157     } else if (operator == Operator.Or) {
158         if (paramType != ParameterType.None) {
159             revert UnsuitableParameterType(index);
160         }
161     } else if (operator == Operator.Nor) {
162         if (paramType != ParameterType.None) {
163             revert UnsuitableParameterType(index);
164         }
165     } else if (operator == Operator.Xor) {
166         if (paramType != ParameterType.None) {
```

```
167             revert UnsuitableParameterType(index);
168         }
169     } else if (operator == Operator.Matches) {
170         if (
171             paramType != ParameterType.Tuple &&
172             paramType != ParameterType.Array &&
173             paramType != ParameterType.AbiEncoded
174         ) {
175             revert UnsuitableParameterType(index);
176         }
177     } else if (operator == Operator.ArraySome) {
178         if (paramType != ParameterType.Array) {
179             revert UnsuitableParameterType(index);
180         }
181     } else if (operator == Operator.ArrayEvery) {
182         if (paramType != ParameterType.Array) {
183             revert UnsuitableParameterType(index);
184         }
185     } else if (operator == Operator.ArraySubset) {
186         if (paramType != ParameterType.Array) {
187             revert UnsuitableParameterType(index);
188         }
189     } else if (operator == Operator.EqualToAvatar) {
190         if (paramType != ParameterType.Static) {
191             revert UnsuitableParameterType(index);
192         }
193     } else if (operator == Operator.EqualTo) {
194         if (
195             paramType != ParameterType.Static &&
196             paramType != ParameterType.Dynamic &&
197             paramType != ParameterType.Tuple &&
198             paramType != ParameterType.Array
199         ) {
200             revert UnsuitableParameterType(index);
201         }
202         if (compValue.length % 32 != 0) {
203             revert UnsuitableCompValue(index);
204         }
205     } else if (operator == Operator.GreaterThan) {
206         if (paramType != ParameterType.Static) {
207             revert UnsuitableParameterType(index);
208         }
209         if (compValue.length != 32) {
210             revert UnsuitableCompValue(index);
211         }
212     } else if (operator == Operator.LessThan) {
213         if (paramType != ParameterType.Static) {
214             revert UnsuitableParameterType(index);
```

```
215         }
216         if (compValue.length != 32) {
217             revert UnsuitableCompValue(index);
218         }
219     } else if (operator == Operator.SignedIntGreaterThan) {
220         if (paramType != ParameterType.Static) {
221             revert UnsuitableParameterType(index);
222         }
223         if (compValue.length != 32) {
224             revert UnsuitableCompValue(index);
225         }
226     } else if (operator == Operator.SignedIntLessThan) {
227         if (paramType != ParameterType.Static) {
228             revert UnsuitableParameterType(index);
229         }
230         if (compValue.length != 32) {
231             revert UnsuitableCompValue(index);
232         }
233     } else if (operator == Operator.Bitmask) {
234         if (
235             paramType != ParameterType.Static &&
236             paramType != ParameterType.Dynamic
237         ) {
238             revert UnsuitableParameterType(index);
239         }
240         if (compValue.length != 32) {
241             revert MalformedBitmask(index);
242         }
243     } else if (operator == Operator.Custom) {
244         if (
245             paramType != ParameterType.Static &&
246             paramType != ParameterType.Dynamic &&
247             paramType != ParameterType.Tuple &&
248             paramType != ParameterType.Array
249         ) {
250             revert UnsuitableParameterType(index);
251         }
252     } else if (operator == Operator.WithinAllowance) {
253         if (paramType != ParameterType.Static) {
254             revert UnsuitableParameterType(index);
255         }
256     } else if (operator == Operator.EtherWithinAllowance) {
257         if (paramType != ParameterType.None) {
258             revert UnsuitableParameterType(index);
259         }
260     } else if (operator == Operator.CallWithinAllowance) {
261         if (paramType != ParameterType.None) {
262             revert UnsuitableParameterType(index);
```

```
263             }
264         }
265 }
```

Recommendation:

We advise them to be grouped into a single conditional, greatly increasing the legibility of the `Integrity::_content` function as well as optimizing its execution cost.

Alleviation:

Operator group types are properly grouped in the validation process within `Integrity::_node`, optimizing the codebase's bytecode significantly.

IYT-02C: Loop Iterator Optimizations

Type	Severity	Location
Gas Optimization	Informational	Integrity.sol:L35, L62, L68, L84, L102, L119, L278

Description:

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

Example:

packages/evm/contracts/Integrity.sol

SOL

```sol
35  for (uint256 i = 0; i < conditions.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:

The Gnosis Guild team considered this exhibit but has opted not to apply it to avoid broad usage of the `unchecked` paradigm. As such, we consider this exhibit acknowledged.

IYT-03C: Redundant Validation of Sub-Type Tree

Type	Severity	Location
Gas Optimization	Informational	Integrity.sol:L95-L96

Description:

The `Integrity::compatibleSubTypeTree` is invoked unconditionally for logical expressions (i.e. `Operator.And`), however, they may contain a single child in certain cases rendering the validation redundant.

Example:

packages/evm/contracts/Integrity.sol

SOL

```
84  for (uint256 i = 0; i < conditions.length; i++) {
85      ConditionFlat memory condition = conditions[i];
86      if (
87          (condition.operator >= Operator.And &&
88              condition.operator <= Operator.Xor)
89      ) {
90          // must have no compValue
91          if (condition.compValue.length != 0) {
92              revert UnsuitableCompValue(i);
93          }
94
95          // must have at least one child
96          if (childrenBounds[i].length == 0) {
97              revert UnsuitableChildrenCount(i);
98          }
99      }
100 }
101
102 for (uint256 i = 0; i < conditions.length; i++) {
103     ConditionFlat memory condition = conditions[i];
104     if (
105         (condition.operator >= Operator.And &&
106             condition.operator <= Operator.Xor)
107     ) {
108         compatibleSubTypeTree(conditions, i, childrenBounds);
```

```
109     }
110
111     if (
112         (condition.paramType == ParameterType.Array &&
113             childrenBounds[i].length > 1)
114     ) {
115         compatibleSubTypeTree(conditions, i, childrenBounds);
116     }
117 }
```

Recommendation:

We advise the same `childrenBounds` conditional of the **`Integrity::compatibleSubTypeTree`** invocation for `Array` parameter types to be assimilated into the referenced `if` clause, ensuring that the **`Integrity::compatibleSubTypeTree`** function is invoked optimally.

Alleviation:

The code of **`Integrity::_topology`** (now renamed to `Integrity::_tree`) has been refactored to no longer contain this inefficiency, validating `childrenBounds` lengths on a need-to basis and thus optimizing the validation process.

MultiSendUnwrapper Code Style Findings

MSU-01C: Deprecated `if-revert` Pattern

Type	Severity	Location
Code Style	Informational	MultiSendUnwrapper.sol:L16-L17, L19-L20, L29-L30, L35-L36

Description:

The referenced statements utilize the `if-revert` pattern without any custom error messages.

Example:

packages/evm/contracts/adapters/MultiSendUnwrapper.sol

SOL

```sol
16  if (value != 0) {
17      revert();
18  }
19  if (operation != Enum.Operation.DelegateCall) {
20      revert();
21  }
```

Recommendation:

We advise either custom error messages to be defined for the referenced failure cases or the statement blocks to be replaced by `require` statements with a descriptive error message, either of which we consider an adequate resolution to this exhibit.

Alleviation:

All `revert` statements in the codebase have been updated to yield one of the three newly declared custom errors in the `MultiSendUnwrapper` contract, standardizing the code's style and enhancing its debugging capabilities.

MSU-02C: Non-Standard Definition of Function Signature

Type	Severity	Location
Code Style	Informational	MultiSendUnwrapper.sol:L7

Description:

The referenced `bytes4` literal is meant to represent the function signature of `multiSend(bytes)`, however, it is defined in its literal format.

Example:

packages/evm/contracts/adapters/MultiSendUnwrapper.sol

SOL

```
7   bytes4 private constant SELECTOR = 0x8d80ff0a;
```

Recommendation:

We advise an `interface` to be defined that represents the `IMultiSend` contract with the correct function signature. Consequently, the function signature's selector can be accessed via the homonym syntax (i.e. `IMultiSend.multiSend.selector`) greatly optimizing the code's legibility.

Alleviation:

Our recommendation was applied to the letter, declaring an `IMultiSend` interface in `Types` and utilizing the `selector` syntax in place of a function signature literal where `SELECTOR` was utilized.

Packer Code Style Findings

PRE-01C: Loop Iterator Optimizations

Type	Severity	Location
Gas Optimization	Informational	Packer.sol:L23, L50

Description:

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

Example:

packages/evm/contracts/packers/Packer.sol

SOL

```sol
23  for (uint256 i; i < count; ++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:

The Gnosis Guild team considered this exhibit but has opted not to apply it to avoid broad usage of the `unchecked` paradigm. As such, we consider this exhibit acknowledged.

PRE-02C: Optimization of Array Iteration

Type	Severity	Location
Gas Optimization	Informational	Packer.sol:L83-L85

Description:

As the `Integrity::enforce` function guarantees that the `parent` entries of the `conditions` array are defined in ascending order, it is possible to optimize the referenced comparison.

Example:

packages/evm/contracts/packers/Packer.sol

SOL

```sol
} else {
    uint256 length = conditions.length;
    unchecked {
        for (uint256 j = index + 1; j < length; ++j) {
            if (conditions[j].parent != index) {
                continue;
            }
            if (!_isInline(conditions, j)) {
                return false;
            }
        }
    }
    return true;
}
```

Recommendation:

We advise the code to perform a `continue` operation when `conditions[j].parent < index` and a `break` operation when `conditions[j].parent > index`, significantly optimizing the `Packer::_isInline` function's gas cost.

Alleviation:

The code was optimized per our recommendation, performing a `continue` or `break` operation based on the `index` by taking advantage of the `conditions` array's ascending order.

Periphery Code Style Findings

PYR-01C: Redundant Code Duplication

Type	Severity	Location
Code Style	Informational	Periphery.sol:L26, L34

Description:

The referenced statements are replicated within the `Core::_key` function.

Example:

packages/evm/contracts/Periphery.sol

SOL

```sol
12  abstract contract Periphery is OwnableUpgradeable {
13      event SetUnwrapAdapter(
14          address to,
15          bytes4 selector,
16          ITransactionUnwrapper adapter
17      );
18
19      mapping(bytes32 => ITransactionUnwrapper) public unwrappers;
20
21      function setTransactionUnwrapper(
22          address to,
23          bytes4 selector,
24          ITransactionUnwrapper adapter
25      ) external onlyOwner {
26          unwrappers[bytes32(bytes20(to)) | (bytes32(selector) >> 160)] = adapter;
27          emit SetUnwrapAdapter(to, selector, adapter);
28      }
29
30      function getTransactionUnwrapper(
31          address to,
32          bytes4 selector
33      ) internal view returns (ITransactionUnwrapper) {
34          return unwrappers[bytes32(bytes20(to)) | (bytes32(selector) >> 160)];
35      }
```

```
36  }
```

Recommendation:

We advise the relevant function to either be imported and utilized within `Periphery` or its logic to be relocated to a common dependency in use by both contracts, either of which we consider an adequate resolution to this exhibit.

Alleviation:

The Gnosis Guild team evaluated this exhibit and has opted to retain the code inline instead of refactoring it to a shared dependency. As such, we consider this exhibit acknowledged.

PermissionBuilder Code Style Findings

PBR-01C: Suboptimal Struct Declaration Styles

Type	Severity	Location
Code Style	Informational	PermissionBuilder.sol:L63-L66, L77-L80, L91-L94

Description:

The linked declaration styles of the referenced structs are using index-based argument initialization.

Example:

packages/evm/contracts/PermissionBuilder.sol

SOL

```sol
roles[roleKey].targets[targetAddress] = TargetAddress(
    Clearance.Target,
    options
);
```

Recommendation:

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

Alleviation:

The three referenced `struct` declarations have been adjusted to use the key-value declaration style, greatly optimizing their legibility.

PermissionChecker Code Style Findings

PCR-01C: Inefficient Iteration of Child Conditions

Type	Severity	Location
Gas Optimization	Informational	PermissionChecker.sol:L423-L426, L431

Description:

The `PermissionChecker::_xor` function will inefficiently loop through all remaining child conditions if more than one validate as `Status.Ok`.

Example:

packages/evm/contracts/PermissionChecker.sol

SOL

```sol
406 function _xor(
407     uint256 value,
408     bytes calldata data,
409     Condition memory condition,
410     ParameterPayload memory payload,
411     Consumption[] memory consumptions
412 ) private pure returns (Status, Result memory result) {
413     uint256 okCount;
414     unchecked {
415         for (uint256 i; i < condition.children.length; ++i) {
416             (Status status, Result memory _result) = _walk(
417                 value,
418                 data,
419                 condition.children[i],
420                 payload,
421                 consumptions
422             );
423             if (status == Status.Ok) {
424                 result = _result;
425                 okCount = okCount + 1;
426             }
427         }
428     }
```

```
429
430     return
431         okCount == 1
432             ? (Status.Ok, result)
433             : (
434                 Status.XorViolation,
435                 Result({consumptions: consumptions, info: 0})
436             );
437 }
```

Recommendation:

We advise the code to instead `break` when a second `Status.Ok` is identified within the referenced `if` clause.

Alleviation:

The code of the **`PermissionChecker::_xor`** function has been omitted as a result of finding `PCR-01M`. As such, we consider this exhibit no longer applicable.

PCR-02C: Optimization of Data Extraction

Type	Severity	Location
Gas Optimization	Informational	PermissionChecker.sol:L597-L602

Description:

The `PermissionChecker::_bitmask` function will inefficiently extract a single `word` from the `data` payload by using the `Decoder::pluck` function instead of the `Decoder::word` counterpart.

Example:

packages/evm/contracts/PermissionChecker.sol

SOL

```
597 bool isInline = condition.paramType == ParameterType.Static;
598 bytes calldata value = Decoder.pluck(
599     data,
600     payload.location + (isInline ? 0 : 32),
601     payload.size - (isInline ? 0 : 32)
602 );
```

Recommendation:

We advise the code to be optimized akin to `PermissionChecker::_compare`, performing a `Decoder::word` operation when the parameter is in-line (`Static`) and a `Decoder::pluck` operation when it is dynamic (`Dynamic`).

Alleviation:

The Gnosis Guild team evaluated this exhibit but opted to retain the current behaviour of the codebase for the sake of code clarity and evident behaviour.

PermissionTracker Code Style Findings

PTR-01C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	PermissionTracker.sol:L74-L75

Description:

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

Example:

packages/evm/contracts/PermissionTracker.sol

SOL

```
74  allowances[key].balance = balance - consumed;
75  allowances[key].refillTimestamp = refillTimestamp;
```

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 referenced `allowances` mapping lookups have been optimized as advised, utilizing a local `storage` pointer and thus performing a single lookup operation for the desired `Allowance` entry.

Roles Code Style Findings

RSE-01C: Inefficient Transfer of Ownership

Type	Severity	Location
Gas Optimization	Informational	Roles.sol:L61

Description:

The referenced statement will transfer the ownership of the contract during its initialization using the `OwnableUpgradeable::transferOwnership` function which is inefficient as it redundantly applies the `onlyOwner` modifier.

Example:

packages/evm/contracts/Roles.sol

SOL

```sol
48  /// @dev There is no zero address check as solidty will check for
49  /// missing arguments and the space of invalid addresses is too large
50  /// to check. Invalid avatar or target address can be reset by owner.
51  function setUp(bytes memory initParams) public override initializer {
52      (address _owner, address _avatar, address _target) = abi.decode(
53          initParams,
54          (address, address, address)
55      );
56      __Ownable_init();
57
58      avatar = _avatar;
59      target = _target;
60
61      transferOwnership(_owner);
62      setupModules();
63
64      emit RolesModSetup(msg.sender, _owner, _avatar, _target);
65  }
```

Recommendation:

We advise its internal counterpart, `OwnableUpgradeable::_tranfserOwnership`, to be utilized instead minimizing the gas cost required during the contract's initialization.

Alleviation:

The underscore prefixed counterpart of the `OwnableUpgradeable::transferOwnership` function is now properly utilized by the code, optimizing its deployment cost by avoiding redundant access control checks.

RSE-02C: Loop Iterator Optimization

Type	Severity	Location
Gas Optimization	Informational	Roles.sol:L79

Description:

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

Example:

packages/evm/contracts/Roles.sol

SOL

```
79  for (uint16 i; i < roleKeys.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 Gnosis Guild team considered this exhibit but has opted not to apply it to avoid broad usage of the `unchecked` paradigm. As such, we consider this exhibit acknowledged.

Topology Code Style Findings

TYG-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	Topology.sol:L118

Description:

The linked mathematical operation is guaranteed to be performed safely by surrounding conditionals evaluated in either `require` checks or `if-else` constructs.

Example:

packages/evm/contracts/Topology.sol

SOL

```
117 for (uint256 i = start; i < end; ++i) {
118     result.children[i - start] = typeTree(conditions, i, bounds);
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statement to be wrapped in an `unchecked` code block thereby optimizing its execution cost.

Alleviation:

The Gnosis Guild team considered this exhibit but has opted not to apply it to avoid broad usage of the `unchecked` paradigm. As such, we consider this exhibit acknowledged.

TYG-02C: Loop Iterator Optimization

Type	Severity	Location
Gas Optimization	Informational	Topology.sol:L117

Description:

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

Example:

packages/evm/contracts/Topology.sol

SOL

```sol
117 for (uint256 i = start; i < end; ++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 loop iterator increment statement has been relocated to the end of the `for` loop body and wrapped in an `unchecked` code block, optimizing its execution.

Types Code Style Findings

TSP-01C: Incorrect Grouping of `EqualToAvatar` Operation (Documentation)

Type	Severity	Location
Code Style	Informational	Types.sol:L49

Description:

The `Operator.EqualToAvatar` operator is a unique operator as it is not expected to have children nor a `compValue`.

Example:

packages/evm/contracts/Types.sol

SOL

```
35  // 05-16: COMPLEX EXPRESSIONS
36  //          paramType: AbiEncoded / Tuple / Array,
37  //          ✅ children
38  //          🚫 compValue
39  /* 05: */ Matches,
40  /* 06: */ ArraySome,
41  /* 07: */ ArrayEvery,
42  /* 08: */ ArraySubset,
43  /* 09: */ _Placeholder09,
44  /* 10: */ _Placeholder10,
45  /* 11: */ _Placeholder11,
46  /* 12: */ _Placeholder12,
47  /* 13: */ _Placeholder13,
48  /* 14: */ _Placeholder14,
49  /* 15: */ EqualToAvatar,
```

Recommendation:

We advise it to be documented properly, ensuring that the mandatory `children` trait is not inferred by the enum's documentation incorrectly.

Alleviation:

The `EqualToAvatar` operator has been set to its dedicated "SPECIAL COMPARISON" category, clearly denoting the expected `children` and `compValue` entries for such operator types.

TSP-02C: Suboptimal Documentation of Enum

Type	Severity	Location
Standard Conformity	Informational	Types.sol:L10-L17

Description:

The `ParameterType` enum is pivotal to the role modifier's operation of the Gnosis Zodiac ecosystem as it is assumed to always occupy at most 3 bits (i.e. up to `0x07`), a trait that is not necessarily enforced.

Example:

packages/evm/contracts/Types.sol

SOL

```sol
4   /**
5    * @title Types - a file that contains all of the type definitions used throughout
6    * the Zodiac Roles Mod.
7    * @author Cristóvão Honorato - <cristovao.honorato@gnosis.io>
8    * @author Jan-Felix Schwarz  - <jan-felix.schwarz@gnosis.io>
9    */
10  enum ParameterType {
11      None,
12      Static,
13      Dynamic,
14      Tuple,
15      Array,
16      AbiEncoded
17  }
```

Recommendation:

We advise the documentation of the `ParameterType` enum to be expanded, annotating that the `enum` can introduce at most two more data types as otherwise the `BufferPacker` function will fail to execute properly. As an alternative, the two remaining parameter slots can be declared similarly to `Operator` as placeholders.

Alleviation:

The Gnosis Guild team evaluated this exhibit but has opted to not take any action. As such, we consider this exhibit acknowledged.

Types Code Style Findings

TSE-01C: Multiple Top-Level Declarations

Type	Severity	Location
Code Style	Informational	Types.sol:L6, L15, L24

Description:

The referenced file contains multiple top-level declarations that decrease the legibility of the codebase.

Example:

packages/evm/contracts/adapters/Types.sol

SOL

```sol
6   struct UnwrappedTransaction {
7       Enum.Operation operation;
8       address to;
9       uint256 value;
10      // We wanna deal in calldata slices. We return location, let invoker slice
11      uint256 dataLocation;
12      uint256 dataSize;
13  }
14
15  interface ITransactionUnwrapper {
16      function unwrap(
17          address to,
18          uint256 value,
19          bytes calldata data,
20          Enum.Operation operation
21      ) external view returns (UnwrappedTransaction[] memory result);
22  }
23
24  interface ICustomCondition {
```

Recommendation:

We advise all highlighted top-level declarations to be split into their respective code files, avoiding

unnecessary imports as well as increasing the legibility of the codebase.

Alleviation:

The Gnosis Guild team evaluated this exhibit and has opted to not apply a remediation for it in the current iteration of the codebase.

WriteOnce Code Style Findings

WOE-01C: Multiple Top-Level Declarations

Type	Severity	Location
Code Style	Informational	WriteOnce.sol:L4, L11

Description:

The referenced file contains multiple top-level declarations that decrease the legibility of the codebase.

Example:

packages/evm/contracts/WriteOnce.sol

SOL

```sol
interface ISingletonFactory {
    function deploy(
        bytes memory initCode,
        bytes32 salt
    ) external returns (address);
}

library WriteOnce {
```

Recommendation:

We advise all highlighted top-level declarations to be split into their respective code files, avoiding unnecessary imports as well as increasing the legibility of the codebase.

Alleviation:

The Gnosis Guild team evaluated this exhibit and has opted to not apply a remediation for it in the current iteration of the codebase.

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.

External Call Validation

Many contracts that interact with DeFi contain a set of complex external call executions that need to happen in a particular sequence and whose execution is usually taken for granted whereby it is not always the case. External calls should always be validated, either in the form of `require` checks imposed at the contract-level or via more intricate mechanisms such as invoking an external getter-variable and ensuring that it has been properly updated.

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 Solidity language boasts that discerns it from other conventional programming languages. For example, the EVM is a 256-bit machine meaning that operations on less-than-256-bit types are more costly for the EVM in terms of gas costs, meaning that loops utilizing a `uint8` variable because their limit will never exceed the 8-bit range actually cost more than redundantly using a `uint256` variable.

Code Style

An official Solidity style guide exists that is constantly under development and is adjusted on each new Solidity release, designating how the overall look and feel of a codebase should be. 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 contract-level variable that is present in the inheritance chain of the local execution level's context.

Gas Optimization

Gas optimization findings relate to ways the codebase can be optimized to reduce the gas cost involved with interacting with it to various degrees. These types of findings are completely optional and are pointed out for the benefit of the project's developers.

Standard Conformity

These types of findings relate to incompatibility between a particular standard's implementation and the project's implementation, oftentimes causing significant issues in the usability of the contracts.

Mathematical Operations

In Solidity, math generally behaves differently than other programming languages due to the constraints of the EVM. A prime example of this difference is the truncation of values during a division which in turn leads to loss of precision and can cause systems to behave incorrectly when dealing with percentages and proportion calculations.

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.

Centralization Concern

This category covers all findings that relate to a significant degree of centralization present in the project and as such the potential of a Single-Point-of-Failure (SPoF) for the project that we urge them to re-consider and potentially omit.

Reentrant Call

This category relates to findings that arise from re-entrant external calls (such as EIP-721 minting operations) and revolve around the inapplicacy of the Checks-Effects-Interactions (CEI) pattern, a pattern that dictates checks (`require` statements etc.) should occur before effects (local storage updates) and interactions (external calls) should be performed last.

Disclaimer

The following disclaimer applies to all versions of the audit report produced (preliminary / public / private) and is in effect for all past, current, and future audit reports that are produced and hosted under Omniscia:

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.